NAVAL POSTGRADUATE SCHOOL
Monterey, California

19980416 114

THESIS

IMPLEMENTATION AND EVALUATION OF AN INS SYSTEM FOR THE SHEPHERD ROTARY VEHICLE

DTIC QUALITY INSPECTED 4

by

Thorsten Leonardy

December, 1997

Advisor:	Xiaoping Yun
Second Reader:	Xavier K. Maruyama

Approved for public release; Distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, Va 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1997	Master's Thesis

4. TITLE AND SUBTITLE
IMPLEMENTATION AND EVALUATION OF AN INS SYSTEM FOR THE SHEPHERD ROTARY VEHICLE

5. FUNDING NUMBERS

6. AUTHORS
Leonardy, Thorsten

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, California 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT(*maximum 200 words*)

An autonomous vehicle must be able to determine its global position even in the absence of external information input. To obtain reliable position information, this would require the integration of multiple navigation sensors and the optimal fusion of the navigation data provided by them.

The approach taken in this thesis was to implement two navigation sensors for a four-wheel drive and steer autonomous vehicle: An inertial measurement unit providing linear acceleration in three dimensions and angular velocity for the vehicle's global motion and shaft encoders providing local motion parameters. An inertial measurement unit is integrated with the Shepherd mobile robot and data acquisition and processing software is developed. Position estimation based on shaft encoder readings is implemented. The framework for future analysis including most general motion profiles have been laid.

The sensor's system performance was evaluated using three different linear motion profiles. Test results indicate that the shaft encoder provide a positioning accuracy better than 99% (typ. 7.5 mm for 1 m motion) under no slip conditions for pure translational motion. The IMU still requires further improvement to allow for both sensors to be combined to an integrated system.

14. SUBJECT TERMS
Robotics, Sensors, Navigation, NPS, Shepherd, Rotary Vehicle

15. NUMBER OF PAGES
114

16. PRICE CODE
n/a

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
239-102

Approved for public release; distribution is unlimited

IMPLEMENTATION AND EVALUATION OF AN INS SYSTEM FOR THE SHEPHERD ROTARY VEHICLE

Thorsten Leonardy
Lieutenant, German Navy
Dipl.-Ing. Nachrichtentechnik, German Armed Forces University, Munich 1989

Submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Physics

from the

NAVAL POSTGRADUATE SCHOOL

December 1997

Author: ____________________
Thorsten Leonardy

Approved by: ____________________
Xiaoping Yun, Thesis Advisor

____
Xavier K. Maruyama, Second Reader

____
William B. Maier, Chairman
Department of Physics

ABSTRACT

An autonomous vehicle must be able to determine its global position even in the absence of external information input. To obtain reliable position information, this would require the integration of multiple navigation sensors and the optimal fusion of the navigation data provided by them.

The approach taken in this thesis was to implement two navigation sensors for a four-wheel drive and steer autonomous vehicle: An inertial measurement unit providing linear acceleration in three dimensions and angular velocity for the vehicle's global motion and shaft encoders providing local motion parameters. An inertial measurement unit is integrated with the Shepherd mobile robot and data acquisition and processing software is developed. Position estimation based on shaft encoder readings is implemented. The framework for future analysis including most general motion profiles have been laid.

The sensor's system performance was evaluated using three different linear motion profiles. Test results indicate that the shaft encoder provide a positioning accuracy better than 99% (typ. 7.5 mm for 1 m motion) under no slip conditions for pure translational motion. The IMU still requires further improvement to allow for both sensors to be combined to an integrated system.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

This research was only possible due to the efforts of many people. Instrumental in the success of this research was the assistance of Michael Williams. His expertise in wide-ranging technical experience and software engineering skills enabled the development of the system up to this point.

Many thanks to my thesis advisor Dr. Xiaping Yun and second reader Dr. Xavier K. Maruyama for their knowledge of, guidance with, and enthusiasm for the subject material. Both made the conduction of this thesis an enjoyable and rewarding experience.

I owe debt of gratitude to the faculty and staff in the Physics Department and Combat Systems Curriculum for the outstanding education that was made available to my fellow officers and I through our course of study. Special thanks goes to Dr. William B. Maier who guided me through my very first and final quarter at the Naval Postgraduate School. Dr. Armstead added a very entertaining note to the endeavors of teaching Quantum Physics and Statistical Physics.

Appreciation goes to the founders of the TEX and LATEX word-processing system, Donald Knuth and Lesley Lamport. It amazes me that such a capable scientific typesetting software is freely available in the public domain. I certainly enjoyed writing this document in LATEX2e.

Most importantly, I would like to thank my wife Ute. She endured many extended periods of my absence without complaint. This effort would not have been possible without her. Thanks for your love and patience!

I. INTRODUCTION

A. BACKGROUND AND MOTIVATION

Landmines have become an ever increasing threat for the civilian communities in post-war scenarios. Several million land mines are scattered around the world annually causing more than 10,000 fatalities and more than 20,000 severe injuries to non-combattants.

Since effective multi-national proliferation treaties banning the use of anti-personnel mines are not yet in place and with major producers for those mines not likely to sign these treaties because of their important impact on conventional warfare, it is essential to develop and deploy equipment for detection of anti-personnel mines in mine-contaminated regions.

Moreover, many countries are downsizing their armed forces due to budget constraints and thus turning over formerly used defense sites to the local communities. Wide areas of these defense sites (such as proving ground, rifle ranges, ...) are contaminated with unexploded ordnance (UXO). The contaminated land must be cleared before transferring to civilian use.

B. OBJECTIVE

At present, there are not many effective means for mine and UXO detection available. The current approach to mine and UXO detection and clearance is labor and time intensive and dangerous: explosive ordnance disposal (EOD) personnel walks slowly over the area that is to be cleared, trying to detect buried, half buried or totally exposed material. Once an object is found, successive steps in the clearance process would include:

- detect,
- identify,
- excavate,
- defuse,
- transport

and

- dispose

the object in question. It is therefore desireable to develop a robust, low-cost tool for persuing the above steps through the use of robotics and advanced sensing techniques meeting the following requirements:

- Robustness for operation in rough terrain
- Expandability for different sensors and equipment
- Precise navigation tools

Multi-disciplinary research conducted in the Departments of Electrical and Computer Engineering, Computer Science, Aeronautics and Astronautics, and the Physics Department at the Naval Postgraduate School, centers around the development of a semi-autonomous robot system for land mine/UXO searching/processing tasks in humanitarian operations [2]. This project has required the cooperative effort of several NPS thesis. The emphasis of this thesis is the implementation of an integrated navigation system. In the long term, the system components will be comprised by a land vehicle, an aerial vehicle, and a ground-based control center.

The land vehicle, specifically designed for the aforementioned tasks is four-wheel steerable and drivable. A prototype vehicle called SHEPHERD is currently in use for this research project. The unique design of SHEPHERD provides a high level of sophistication for motion control for it to be able to precisely traverse rough terrain. The interested reader is referred to [1]. The scope of this project, in general, is very comprehensive and encompasses many scientific areas. In particular, interdisciplinary tools such as physics principles including coordinate transformations, kinematics and mechanics of rigid bodies, and electrical and software engineering tools are used, discussed and covered in this thesis.

In order to control the vehicle and implement efficient search patterns while at the same time reducing redundant search paths, precise knowledge of the vehicle's velocity and position is essential. Using an on-board inertial navigation system, the vehicle's acceleration can be measured and it's 3D motion precisely computed by the on-board computer. However, an inertial sensor alone can provide accurate position information only in the short term, but must be integrated with additional sensors if precise long term positional data is required. The vehicle's rough operation environment makes it essential that extremeley accurate position information is obtained. To meet this requirement, a Global Positioning System (GPS) receiver shall be integrated.

The purpose of this thesis is to implement and evaluate an integrated navigation system for SHEPHERD enabling the operation of the vehicle under extremely rough conditions while at the same time providing accurate position information. This thesis will examine the following research questions:

1. Provide the theoretical background for coordinate transformations,
2. Implement the hardware and software for an Inertial Measurement Unit (IMU),
3. Implement the software to determine position based on the on-board shaft encoders,

4. Develop a scheme for sensor fusion for slip-detection.

C. ORGANIZATION

First, a brief overview of the computer architecture for the Shepherd Rotary Vehicle is given in Chapter II. Secondly, Chapter III defines the basic reference frames that are being used throughout this project. The secondary means of determining the vehicle motion is given by shaft encoders that are used for each of four wheels for both, steering and driving. The software implementation is described in Chapter IV. Chapter V describes the implementation of a low cost inertial measurement system (IMS) both in hardware and software. Its purpose will be to complement the shaft encoder system in situations were slip occurs. How both systems may be unified for slip detection and to further improve the performance of the navigation system is investigated in Chapter VI. Finally, the success and limitations of the use of the system described herein is summarized in Chapter VII providing essential results of this research and recommendations for future research in this area.

II. SYSTEM OVERVIEW

In this chapter we will give a brief computer hardware description of the system configuration for the SHEPHERD Rotary Vehicle. This complements the description given by Mays/Reid [1] and is intended to provide the essential information necessary to understand the cross-references to computer components given in the following Chapters.

All mechanical information for the mobile platform is extensively discussed by Mays/Reid [1]. However, we shall note at this point that the Shepherd Rotary Vehicle is a four wheel drive and steer mobile robot. The four wheels are steerable without limitations and can be rotated and driven in either direction (more than 360 degree of rotation space). The four wheel drive and steer capability shall provide the robustness required for operation in rough terrain (e.g., sand dune scenarios, ...). A side view and front view photo taken from SHEPHERD with a digital camera are shown in Figure 2.1 and Figure 2.2, respectively.

In Figure 2.1 we can can see the four suspension boxes for the four wheels, the steel plate that comprises the main support frame for the robot, the inertial measurement sensor mounted upside down below the steel plate, and a joystick that is used to manually steer the robot in the top right-hand corner. In addition, in the rear view photo you can see the Laptop computer, to its left a switchbox for connecting the Laptop to either a CONSOLE or HOST serial port, and to its right the joystick. Another view, shown in Figure 2.3 shows the Laptop placed on the steel plate and behind it the servo control rack and the VMEBus chassis.

The complete hardware architecture is comprised of the TAURUS Single Board Computer [3], a VME-Bus based single board computer with a Motorola MC68040 as main processor and several other on-board processing components and the VME-Bus. At present, this stand alone computer system is expanded with a servo controller unit that interfaces to the four wheels and a 16-channel differential input A/D-Board. Four channels of the A/D-Board are utilized for the inertial measurement unit (IMU) which is discussed in Chapter V. In the future, the system may be expanded with several other sensors through the use of the VME-Bus. Figure 2.4 shows a block diagram of the system configuration for SHEPHERD.

A. TAURUS BOARD

This section describes the TAURUS single-board computer system's main features. The hardware is based on a dual processor platform using Motorola's 68040 as the main processor and

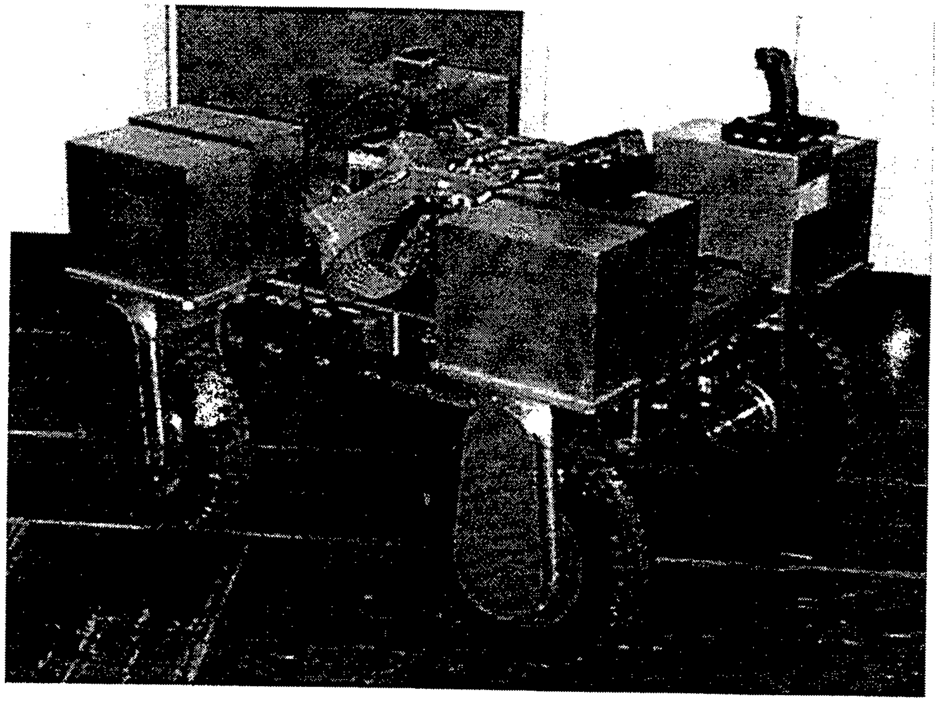


Figure 2.1: Side view from the SHEPHERD Rotary Vehicle.

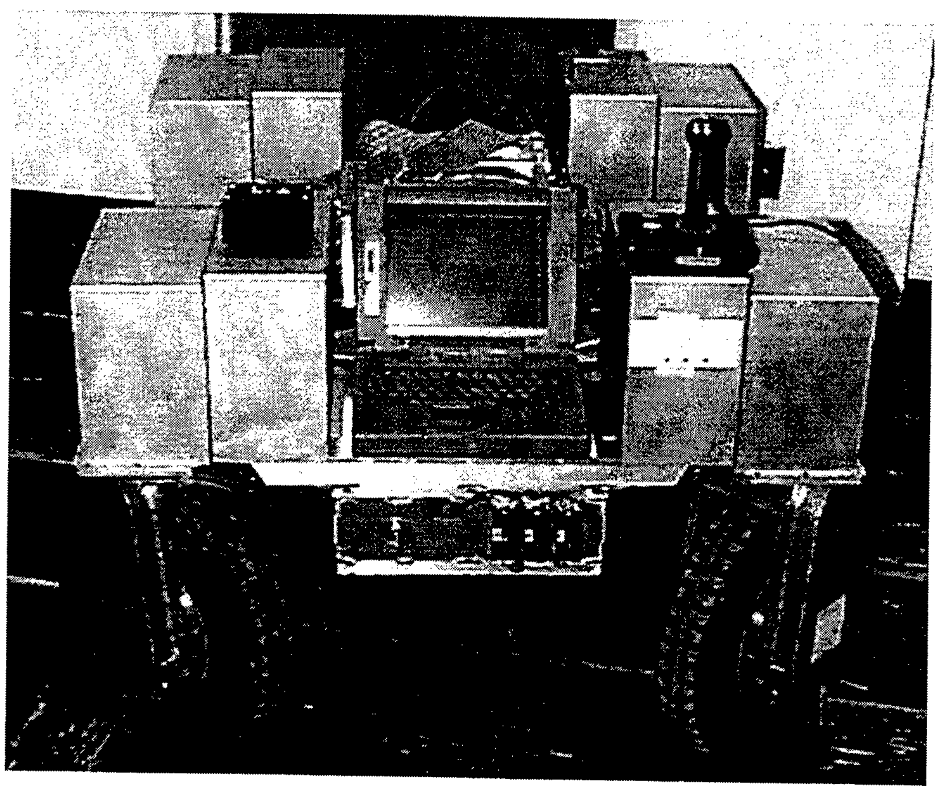


Figure 2.2: Front view from the SHEPHERD Rotary Vehicle with wheels rotated by 45°.

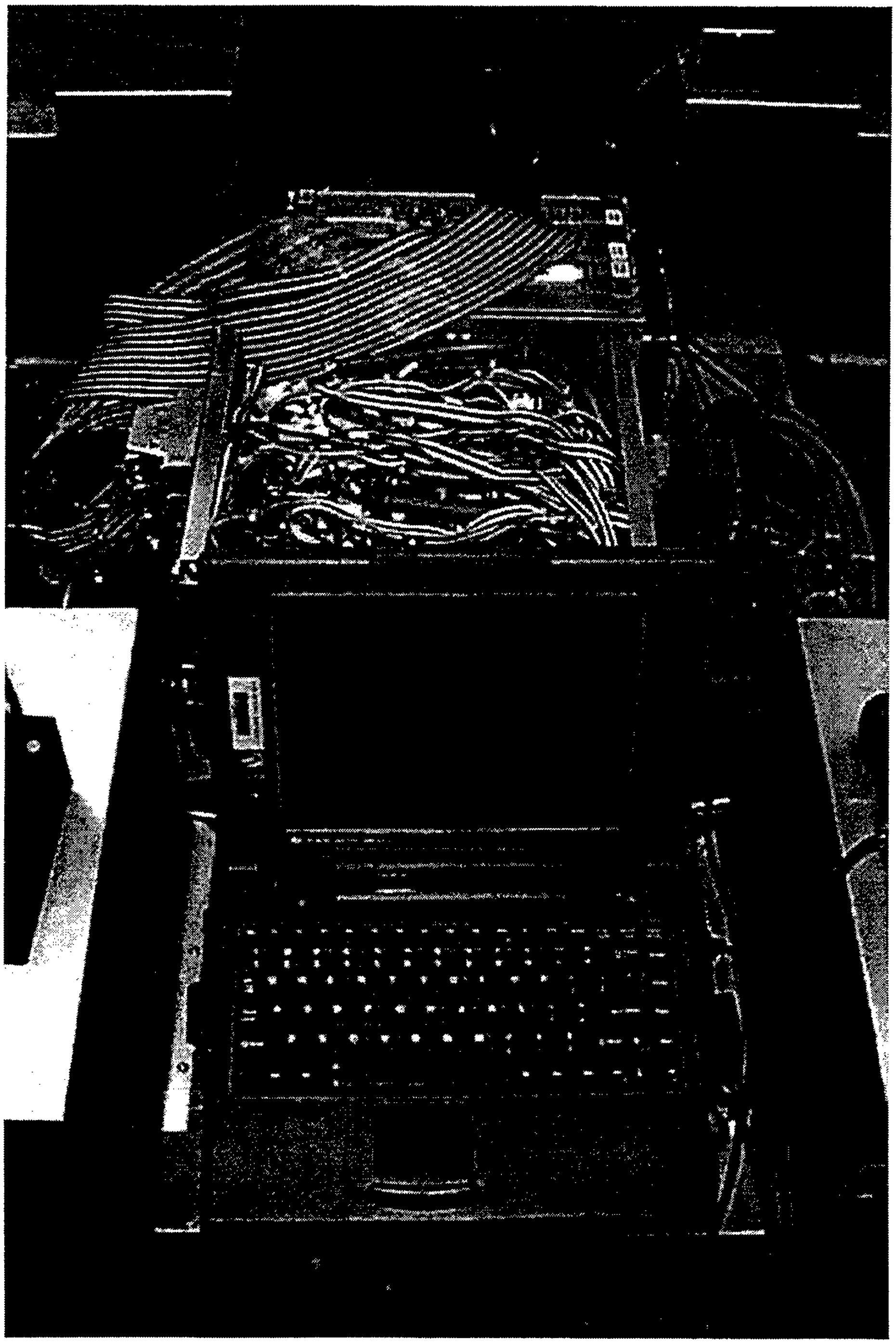


Figure 2.3: Top view from the SHEPHERD Rotary Vehicle. In the front, the Pentium Laptop used as a concole, in the middle the servo controller chassis, and in the back the VMEBus rack.

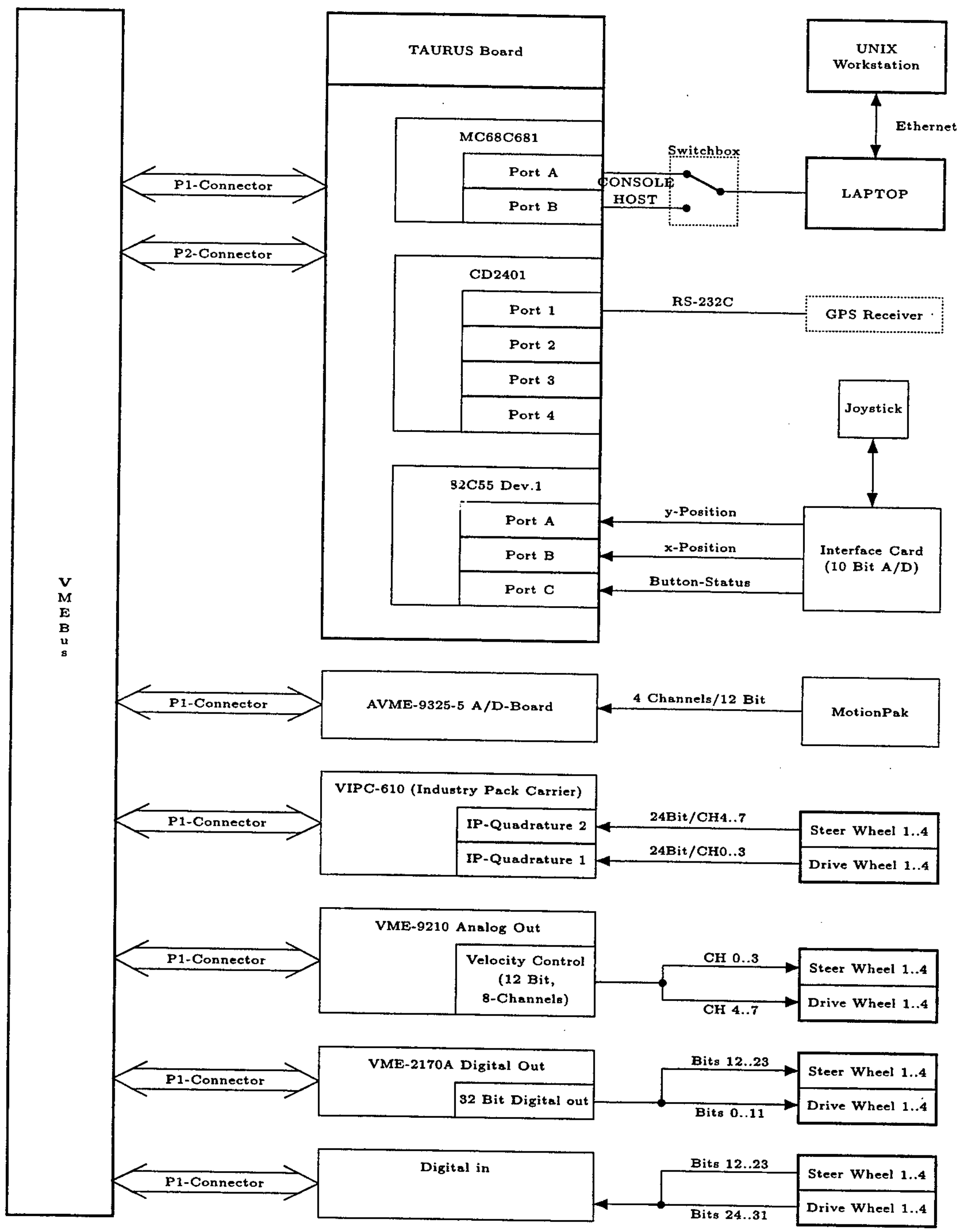


Figure 2.4: Shepherd Rotary Vehicle Hardware Configuration.

the 68030 as a slave processor for basic I/O functions. Signaling between both processors takes place via processor interrupts. The system is attached to a VME bus backplane providing the capability to expand the system as far as main memory and additional sensor devices are concerned. Among the many I/O functions that the TUARUS board provides are:

- six RS-232C serial communication ports (two through a DUART 68C681, and four through a CD2401 Communications Device)
- two 24 bit parallel ports
- several timer/counters: Five provided by the AM9513A System Timing Controller, one timer is provided in the 68C681 serial port device and eight timer/counters are available in the CD2401
- real time calendar clock device MK48T08
- SCSI Port
- Ethernet Port

More information can be obtained from [3] and the respective operating/user manuals for each device. Rather than focussing on all the technological aspects for each device, we merely focus on those important ones for understanding the operation of SHEPHERD.

1. TAURUS Bug Monitor/Debugger

For start-up and debugging/monitoring purposes, a debugger/monitoring program called TAURUSBug resides in the memory region from `0xff800000` through `0xff9fffff` (memory bank 2, see [3], Chapter 2.2). The user may decide whether or not to use this program for the boot-up. However, in the sequel, the project group has made heavy use of the debugging tools provided by TAURUSBug.

2. DUART 68C681

The TAURUS board features a 68C681 device which provides two dual asynchronous receiver/transmitter (DUART) serial ports with RS-232C interface. These two ports are utilized for up-/and downloading of executable code and data and for user interaction with SHEPHERD. Port A is called CONSOLE and Port B is called HOST. Both ports are connected through a switchbox to the laptop computer.

3. Cirrus Logic Communications Controller CD2401

Up to date, only one of the four RS-232C serial ports provided by the Cirrus Logic Communications Controller CD2401 is used for interfacing the GPS receiver.

4. AM9513A Counter/Timer

The AM9513A LSI circuit provides a total of five independent 16-bit timer/counters which can be cascaded to a single 80-Bit timer/counter for long-term observations. The timer number five is used for deriving the motion control clock of T=10 ms: every 10 ms a timer interrupt is issued to trigger another motion control cycle. This 10 ms timer interrupt is clearly the heart of the system. Care should be taken that this interrupt is granted the highest priority level available. This leads to the decision to use timer five instead one of the other four.

5. Programmable Parallel I/O Port Device (Intel 82C55A)

The Taurus board is equipped with two Intel 82C55A devices each providing three 8-Bit wide ports (Port A, B, and C). Only the first device is currently in use for the motion control by means of a joystick. A simple PCB board interfaces an IBM-PC Joystick to this I/O device. However, some minor changes to the layout of the Joystick circuitry had to be made. Port A comprises the x-Position (an 8-bit digital value ranging from 0 ... 255 equivalent to joystick left to right), Port B gives the y-Position in the range 0 ... 255 equivalent to down (`0x00`) and up (`0xff`). Currently, only Bits zero and one are in use from Port C providing status information for the two switches on the throttle (pushing the left switch or the center switch on the trottle will set bit zero and pressing the right button on the throttle will set bit one). The other two push buttons on the left-hand side of the joystick have currently no function. In case that needed, they can easily be connected to any of the six remaining bits of Port C through the PCB board by use of pull-up resistors.

6. Interrupts

Both on-board and off-board Interrupts are supported by the TAURUS board. All on-board Interrupts are routed through the **Interrupt Steering Mechanism (ISM)** to either the 68030 I/O

Processor or via a VMEbus Interface Controller device (VIC068) to the 68040 Processor. Note that an interrupt can only be routed to one processor at a time. The VIC068 guides both, ISM interrupts and VMEbus interrupts to the 68040 processor. This is depicted by Figure 2.5. In accordance with [3], the local interrupts by on-board sources from the ISM to the VIC will be labelled as LIRQ-x whereas the interrupts form the VIC068 to the 68040 processor are labelled IRQ-x.

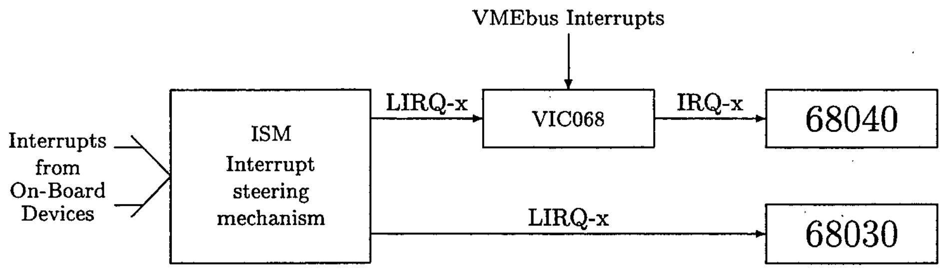


Figure 2.5: Servicing of on-board Interrupts or off-board VME-Bus Interrupts (From Ref. [3])

The ISM combines groups of on-board Interrupts to act as a single interrupt source towards either the 68030 or 68040 processor. It is important to note that the VIC068 device enables the programmer to shift the interrupt levels. In order to handle the proper handshaking in this case, the appropriate LIRQ-Shift-Register in the ISM would have to be set. The TAURUS user's manual [3] p. 2-71 gives the following example:

> ... if LIRQ-5 from the ISM is shifted in the VIC068 to IRQ-3, then the acknowledge signal from the 68040 processor to the VIC068 would be IACK-3 which would be passed on to the ISM device. LIRQ-SR5 (at $FFF4800A - upper nibble) would be set to shift [the] VIC068 IACK-3 input to output ISM-IACK-5.

Some facts that should be remembered:

- each Interrupt group is associated with an ISM Configuration Register Nibble.
- the MSB of each Nibble is the steering Bit, where '0' routes the interrupt to the 68030.
- the remaining three bits of each nibble encode the local interrupt level.
- upon Power-Up or RESET, all On-Board Interrupts are disabled.
- Taurus Vector Table Base address is at `0xffe10xxx` where `xxx` $= 4 \times$ Vector Number.

B. MOTION CONTROL

As indicated in the previous section, a motion control cycle is initiated with every 10 ms timer interrupt. In brief, this motion control cycle is given by the following sequence of logical blocks:

`readEncoder()`	Read all shaft encoders
`computeRates()`	Compute (angular) velocity for all steering and driving motors
`bodyMotion()`	Compute motion parameters for the vehicle's body (bodyMotion)
`wheelMotion()`	Compute the angles and speeds required for each wheel based on the results of bodyMotion
`driveMotors()`	Update the servos for driving and steering motors

The organization of the motion control cycle is described in more detail in Mays/Reid [1]. However, it should be noted that the source code as given there has been modified slightly to make the routines more efficient and thus less time consuming.

III. REFERENCE FRAMES

This chapter gives a brief discussion on reference frames that are being used throughout this thesis.

A. BODY MOTION

In the analysis of the motion of a rigid body, it turns out that considerable simplification in the mathematical formulas for rigid-body motion can be reached if the motion is described with respect to its **principal axes**. The principal axes are chosen such that the cross terms (sometimes called the products of inertia) of the moment of inertia tensor I vanish (see [4] for a more detailed analysis of this). The axes form a right-handed coordinate system with the origin usually taken to be at the body's center of mass (CM). However, at this point we are not concerned with the moment of inertia tensor.

1. Body Reference Frame

For the purpose of describing the kinematics of a body moving on the Earth's surface the reference frame is chosen such that axes of the **body frame**, which we will call frame {B}, are given by the principal axes of the body given as follows:

x - longitudinal axis (oriented from rear to front of body)
y - transversal axis (oriented to the left)
z - normal axis (oriented pointing up, away from the center of the Earth)

2. Sensor Reference Frame

Sensors will be employed with a vehicle in order to measure parameters pertaining to the vehicle's kinematics. The sensor will provide data relative to its own frame, which we will call sensor frame {S}. In general, this frame can be completely different from the body frame. If sensing data is provided in a Cartesian coordinate system, the only difference between {B} and {S} might be an offset (or translational difference) ${}^{B}P_{S,org}$.

3. Earth Reference Frame

In order to express the motion of a body as observed by an outside inertial observer we need to define a suitable inertial reference frame. An inertial reference frame is defined to be the frame for which Newton's laws of motion are valid. For a slow moving vehicle at a particular point on the Earth's surface, a suitable **reference frame** {R} is set up in the following way:

x - pointing north
y - pointing east
z - pointing down, towards the center of the Earth

We will see later in this chapter that the axes x,y and z of this coordinate system refer to the geodetic descriptions of latitude, longitude and geodetic height respectively. Since we do not anticipate any large scale motion (on the order of kilometers) it is sufficient not to concern ourselves with the irregular shape of the Earth and with the resulting mapping/projection problems.

B. GPS SYSTEM

In order to describe both the GPS Satellite motion and receiver motion, it is necessary to choose a common reference system. Most commonly, the motion is described in terms of velocity and position as measured in a Cartesian Coordinate System. The most applicable coordinate system for GPS systems are given as follows: Satellite and GPS receiver motion are described in terms of the Earth-Centered Inertial and Earth-Centered Earth-Fixed coordinate systems respectively. The systems in use are described in detail by Kaplan [5] and are briefly explained below:

1. Earth-Centered Inertial (ECI) Coordinate System

In this system, the origin is the center of mass of the Earth. Satellites orbiting the Earth obey Newton's second law of motion as described in this System. In the ECI system, the xy-plane coincides with the Earth's equatorial plane, the +x-axis points towards some fixed point in space (celestial sphere), the z-axis is taken to be normal to the xy-plane pointing towards the north pole. The set of axis forms a right-handed coordinate system. However, due to the Earth's inhomogeneous shape, irregularities in the Earth's motion cause the ECI frame not to be truly inertial. Therefore, the GPS system defines the ECI reference frame as given by the constellation at 1200 hr UTC on January 1, 2000.

2. Earth-Centered Earth-Fixed (ECEF) Coordinate System

For computing the receivers position, it is more convienient to use a system that is stationary in the earth frame. It is known as Earth-Centered Earth-Fixed (ECEF). As with the ECI frame, the xy-plane is coincident with the Earth's equatorial plane, the x-axis points in the direction of 0° longitude, the y-axis points in the direction of 90° longitude. The x- and y-axes therefore no longer describe fixed directions in inertial space. The z-axis completes the right-handed coordinate system.

3. Conversion between ECI and ECEF

Conversions between ECI and ECEF system are accomplished by means of matrix transformations (rotator matrices) which are not further described in this thesis. It is assumed that the Satellite ephemeris data is already translated into ECEF system.

4. World Geodetic System (WGS-84)

The Department of Defense invented a system to model all irregularities pertaining to describing the Earth's gravitational motion. This system is known as the World Geodetic System (WGS-84). In addition to modeling the gravitational irregularities, the World Geodetic System provides an ellipsoidal model of the Earth. The ECEF coordinate system is affixed to the World Geodetic System reference ellipsoid and thus, latitude, longitude and height of a receiver can be specified with respect to this ellipsoid.

C. TRANSFORMATIONS

To define and manipulate physical quantities such as acceleration, velocity and position we must define coordinate systems and find transformations for describing vectors given in one system with respect to the other. These transformations will be accompanied by conventions for their representation.

A great variety of similar transformations can be found in many textbooks. Not all of them are concisely formulated. It is thus rather confusing to relate different conventions given in different textbooks with each other; even though they may describe the same transformation. A

good introduction on spatial descriptions and transformations is given by [6] and we will therefore briefly outline the most important aspects and conventions as thev pertain to our problem.

The inertial reference frame {R} is given by the set of coordinate axis {x,y,z} where the xy-plane is the plane parallel to the WGS-84 reference ellipsoid (that is, the earth's surface) with x pointing north, y pointing east and z pointing towards the geodetic center of the Earth. The frame {B} which is attached to the body is given by the set of axes { x', y', z' } with x' pointing forward, y' pointing to the left of the body and z' completing the right-handed coordinate system. Figure 3.1 shows both frames.

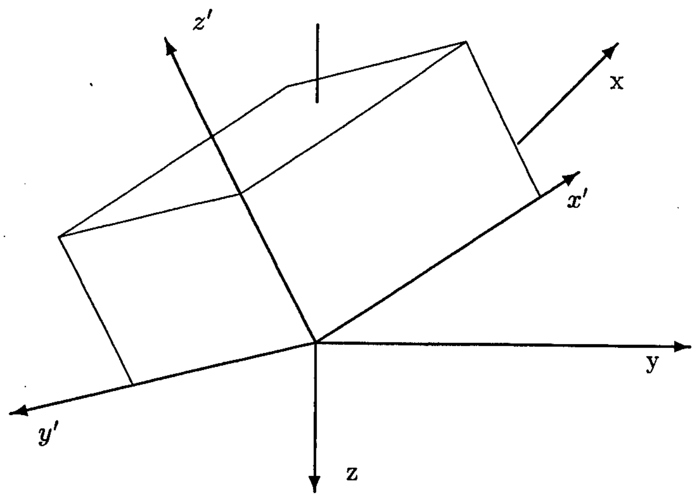


Figure 3.1: Coordinate Frame for Body relative to point on Earth surface. The x/y-plane spans the plane tangent to the Earth's surface.

There are two governing basic methods of representing the orientation of a body (with the Frame {B} attached to it) with respect to the reference frame {R}. One way is to express the principal directions of {B} (unit vectors x', y', z') in terms of the coordinate system {R} and stack these three unit vectors together as the columns of a 3×3 proper orthonormal **rotation matrix**

$$ {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R} = [x'y'z'] $$

where ${}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}$ has the properties that its columns are mutually orthogonal and have unit length and $det({}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}) = 1$. Moreover, it can be shown that the inverse of ${}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}$ is simply its transpose:

$$ {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}^{-1} = {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}^{T} \tag{3.1} $$

and thus giving rise to

$$ {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}^{-1} = {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}^{T} = I \quad . $$

Any vector $\vec{P}$ given with respect to {B} can then be expressed in terms of {R} by the transformation

$$ {}^{\mathrm{R}}\vec{\mathrm{P}} = {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{B}}\vec{\mathrm{P}} $$

Since dealing with 3×3 matrices for describing orientations is usually very tedious, a second way of describing the orientation of a body can be derived from a result from linear algebra. **Cayley's formula** for orthonormal matrices (cited by Craig [6]) states that any 3×3 orthonormal matrix can be specified by just three parameters.

There are many ways to represent orientations with only three parameters. Not all of them are convenient and the reader may be easily confused while looking for those in different textbooks. In the discussion here we follow the conversion of Ref. [6].

1. Roll, Pitch, and Yaw

One way of describing the orientation of a frame {B} relative to the reference frame {R} is by describing the body's orientation by observing successive rotations about the three axes (x,y, and z) of the fixed refernece frame {R}. Craig [6] refers to this convention as **X-Y-Z fixed angles**:

1. start with the frame {B} coincident with the reference frame {R}
2. rotate {B} about ${}^{\mathrm{R}}\vec{\mathrm{x}}$ by the **roll** angle θ
3. rotate {B} about ${}^{\mathrm{R}}\vec{\mathrm{y}}$ by the **pitch** angle ϕ
4. rotate {B} about ${}^{\mathrm{R}}\vec{\mathrm{z}}$ by the **yaw** angle ψ

Each of the three rotations takes place about an axis in the fixed reference frame {R}. The resulting rotation matrix can be obtained by successively rotating the frame {B} about single axes in the stationary frame {R}:

$$ \begin{aligned} {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R} &= {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Z}}(\psi)\,{}^{\mathrm{R}}\mathrm{R}_{\mathrm{Y}}(\phi)\,{}^{\mathrm{R}}\mathrm{R}_{\mathrm{X}}(\theta) \\ &= \begin{bmatrix} \cos(\psi)\cos(\phi) & \cos(\psi)\sin(\phi)\sin(\theta) - \sin(\psi)\cos(\theta) & \cos(\psi)\sin(\phi)\cos(\theta) + \sin(\psi)\sin(\theta) \\ \sin(\psi)\cos(\phi) & \sin(\psi)\sin(\phi)\sin(\theta) + \cos(\psi)\cos(\theta) & \sin(\psi)\sin(\phi)\cos(\theta) - \cos(\psi)\sin(\theta) \\ -\sin(\phi) & \cos(\phi)\sin(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix} \end{aligned} \tag{3.2} $$

where

$$ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{X}}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.3} $$

$$ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Y}}(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{3.4} $$

$$ {}^{R}R_Z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad . \tag{3.5} $$

Therefore, a vector ${}^{B}\vec{a}$ given in frame {B} can be transformed with respect to frame {R} by the transformation

$$ {}^{R}\vec{a} = {}^{R}_{B}\mathbf{R}\, {}^{B}\vec{a} \quad . $$

2. Euler Angles

Another possible description of the frame {B} with respect to frame {R} is given by the **Euler Angles**. As opposed to rotating the frame {B} in successive steps about the fixed axes of {R}, this description will involve successive rotations performed about the principal axes of the rotating frame {B} we are about to move:

1. start with the frame {B} coincident with the reference frame {R}
2. rotate {B} about ${}^{B}\vec{z}$ by the angle ψ
3. rotate {B} about ${}^{B}\vec{y}$ by the angle ϕ
4. rotate {B} about ${}^{B}\vec{x}$ by the angle θ

The resulting rotation matrix is the same as given above in Equation 3.2. Instead of naming the angles θ, ϕ, ψ as roll, pitch, and yaw respectively, they are now being referred to as the Euler Angles. Craig refers to them as the **Z-Y-X Euler Angles**. This transformation is equivalent to the one given by Fossen [7] on page 10 except that we exchanged the naming for roll and pitch ($\theta \leftrightarrow \phi$). The result obtained yields a fundamental statement as given by Craig [6]:

> ... three rotations taken about fixed axis yield the same final orientation as the same three rotations taken in opposite order about the axes of the moving frame.

In this work, we will make reference to the Eulerian angles and this mostly to the fact that the Eulerian angles are easier to recognize. However, the euler angles are equivalent to the roll, yaw and pitch angles.

In this chapter we have laid the framework for transforming vectors from one coordinate system to the other. We will apply this to the Inertial Measurement Unit and develop a scheme for determining the specific acceleration acting on a body even in the presence of the gravitational acceleration.

IV. POSITION DETERMINATION WITH SHAFT ENCODER

This chapter describes the use of the shaft encoders for position determination. It complements and in some cases alters the results obtained by Mays/Reid [1]. As outlined in Mays/Reid [1], each servo motor is equipped with shaft encoders which record the actual angles for all eight motors. This should provide an easy means for direct position determination under the condition that no slip occurs. That is, the difference between an interval T=10 ms by which each encoder (driving and steering) advances is directly proportional to the distance travelled or to the angle each wheel was rotated and accordingly for the time of observation proportional to the linear and angular velocity.

It should be noted that the shaft encoders for the driving motors count positive for a clockwise rotation of the wheel. Thus, if all wheels are driving forward (which implies that wheels 1 and 3 are commanded with negative servo data) the shaft encoder readings will decrease for wheels 2 and 4. In the same manner, if all wheels are steering to the right (clockwise as viewed from above, with negative servo data commanded), the shaft encoder readings will increase for all wheels.

A. DETERMINING THE SERVO PARAMETERS

It might be necessary from time to time to verify and adjust the servo parameters in use for the motion control of SHEPHERD. Therefore, a few test routines have been implemented in the file 'motor.c'. These functions are

`driveTest()`	to determine the driving parameters
`steerTest()`	to determine the steering parameters
`stopTest()`	to determine the interaction between driving and steering for digits commanded to the servos being zero
`velocityTest()`	to obtain a relationship between digits commanded to the driving motors and actual angle rates observed
`circumferenceTest()`	to determine the circumference of the wheels

1. Steer Parameters

For determining the steering parameters the following method has been impemented in function 'steerTest()' in file 'motor.c':

1. align all wheels with hall sensor
2. clear the counters
3. save counter data in variable previous
4. rotate wheels for a certain number of turns and stop time it takes to rotate the wheel
5. read shaft encoder 'current' and compute the counter difference to obtain the rate of turn and number of counts for a turn

The source code is implemented as function 'steerTest()' in the file 'motor.c'. It should be noted that this test should only be conducted for free wheels off the ground, otherwise the vehicle may just wander around.

Some characteristic data corresponding to a specific velocity commanded is shown in Table 4.1. It can be seen from the Table that when steering the wheel, this would interfere with the drive counters as well. The work of Mays/Reid account for this fact by closed loop control. The data was taken for no load applied to the wheels (free turning wheels).

	Wheel 1	Wheel 2	Wheel 3	Wheel 4
count per turn	-92160.2	-92131.7	-92160.3	-92160.1
counts per degree	-256.00	-255.92	-256.00	-256.00
time per turn (sec)	6.97	6.98	6.98	6.98
drive count for turn	2048.0	2047.9	2048.0	2047.9

Table 4.1: Steering Wheel Data at Digits commanded `0x0b00` averaged over 10 turns.

Note when a positive value is commanded to all steering motors that the motion of the wheels as viewed from above is counterclockwise and the shaft encoder readings are negative! From the data, we can derive a relationship between the angular position of the steering motors and the encoder readings

steering wheel 1...4	1 degree = 256 counts
angle turned [radians]	$\theta = 6.8177 \cdot 10^{-5} rad/count$

Table 4.2: Conversion Factor for Steering all Wheels.

The results given above are in agreement with the findings from Mays/Reid [1]. With this data in mind, the angular velocity can be easily measured. All that needs to be done is to record the difference in steer encoder readings for an observation timeframe (T=10ms) and multiply by the above factor and divide by T.

2. Drive Parameters

What is the goal to be determined in this section is: how does the driving data commanded to the drive servos (in the range from -1024 to +1023) relate to the actual driving speed. Moreover, how does driving interfere with the steering, is there any leakage at all? In order to determine this, two functions are in place for use within the SRK.

The function `'driveTest()'` was written in order to determine how the drive encoder readings relate to the angular position of the wheel (if the wheel is viewed as a clock). All this function does is to record the difference in shaft encoder readings for a given number of turns completed. This observation gives rise to the number of counts per degree for driving the wheel. The function does not operate autonomous but rather requires user interaction. The user determines when to start and end the observation period. This procedure was conducted several times at different speeds - although the speed is not of our concern at this point. The results are given in Table 4.3.

driving at speed 0x0800 (1 turn)				
	Wheel 1	Wheel 2	Wheel 3	Wheel 4
count per turn	-102746	-103949	-105340	-104038
counts per degree	-285.41	-288.75	-292.61	-288.99
time per turn (sec)	10.85	10.63	10.97	10.87
drive count for 1 turn	n/a			
driving at speed 0x0800 (averaged over 3 turns)				
	Wheel 1	Wheel 2	Wheel 3	Wheel 4
count per turn	-103989	-104303	-103967	-104229
counts per degree	-288.86	-289.73	-288.80	-298.53
time per turn (sec)	10.85	10.63	10.97	10.87
drive count for 1 turn	n/a			
driving at speed 0x2000 (averaged over 10 turns)				
	Wheel 1	Wheel 2	Wheel 3	Wheel 4
count per turn	-103756	-104143	-104812	-104705
counts per degree	-288.21	-289.29	-291.15	-290.85
time per turn (sec)	2.704	2.698	2.729	2.727
drive count for 1 turn	n/a			
driving at speed 0x2000 (averaged over 100 turns)				
	Wheel 1	Wheel 2	Wheel 3	Wheel 4
count per turn	-104377	-102594	-104440	-104435
counts per degree	-289.92	-284.98	-290.11	-290.10
time per turn (sec)	2.72	2.71	2.72	2.72
drive count for 1 turn	63394.94	63297.88	63331.94	63337.61

Table 4.3: Data obtained for determining drive parameters with program `'driveTest()'`.

It can be seen from the Table that the number of counts per degree for all wheels is given by approximately 290 counts/degree except for wheel two at the commanded speed of 0x0800.

However, it is assumed that the user simply failed in observing the correct number of turns for this wheel. Another test run eventually with even more turns should be conducted. However, for ease of computation and in agreement to Mays/Reid [1], it is expected that for a given number of encoder counts, all wheels will advance by exact the same angle if commanded by the same digit and the conversion is given by

driving wheel 1...4	1 degree = 290 counts
angle driven [radians]	$\theta = 6.018376731 \cdot 10^{-5} rad/count$

Table 4.4: Conversion Factor for Driving all Wheels.

In a second step, a function `'velocityTest()'` was implemented in the source file 'motor.c' in order to determine the driving speed as a function of servo data sent to the driving servos. The inner workings of this function are quite simple:

1. Align all wheels, set `speed = 500`.
2. Set all motors to `speed`.
3. Wait one second to let servos attain steady state.
4. Observe the difference in shaft encoder readings for an observation period of one second. Store the readings in main memory (starting at `0x00100000`) at consecutive locations.
5. Decrease speed = `speed -10`.
6. If speed < -500 stop, otherwise repeat the loop with step 2.
7. Stop the test program.

Once the program was done, the data (steering and driving delta for every second) was downloaded as an ASCII dump to the notebook, converted to decimals and further analyzed using the MATLAB function `'velocity.m'`. Although it was - based on the results from Mays/Reid - expected to obtain a nonlinear relationship between the velocity (which is proportinal to the difference in encoder readings) and the commanded digits, the results proved to be quite different.

For free floating wheels, the drive encoder advances for a given speed during the time interval of 1 sec are shown in Figure 4.1 and the equivalent steer encoder differences are shown in Figure 4.2. To solidify the results, a second experiment, now with the vehicle on the ground has been conducted. The results according to this experiment are shown in Figure 4.3 and Figure 4.4.

As can be seen from the graphs, both experiments show the same linear relationship for the driving of all wheels with just slightly changing parameters and in addition to this, the interaction from driving to steering for each wheel is insignificant and can be neglected. The test was conducted a total of three times, two times with the wheels on the ground and the vehicle moving in a straight

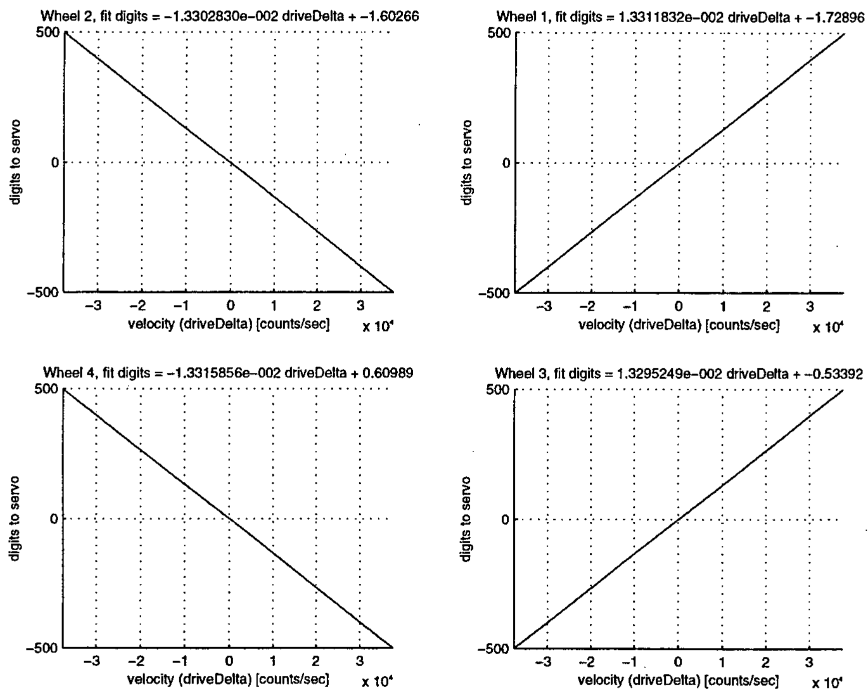


Figure 4.1: Commanded Digits versus Encoder Differences for Free Floating Wheels.

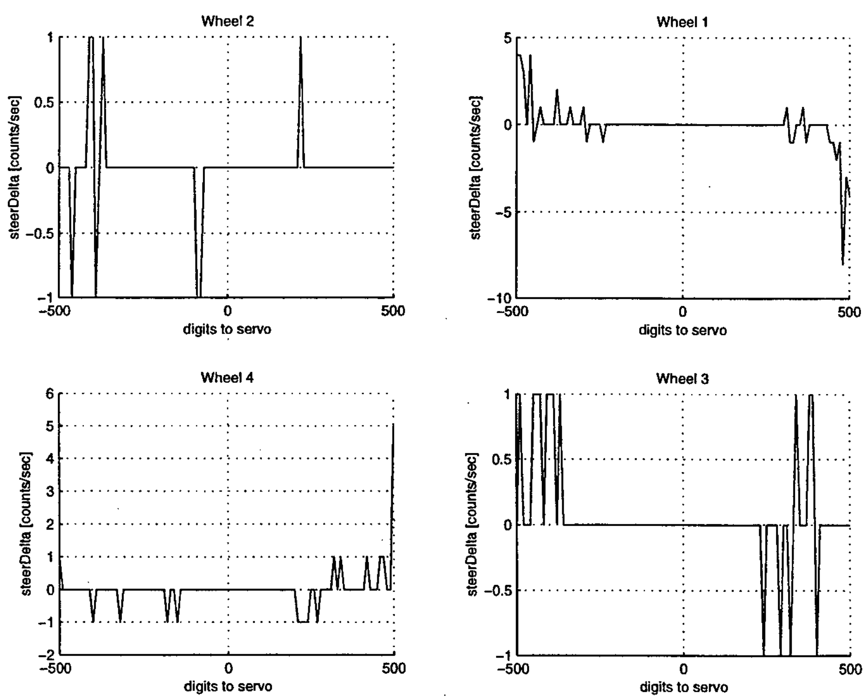


Figure 4.2: Influence of Commanded Drive Digits on Steering Wheels. Plot shows Encoder Differences vs. Commanded Drive Digits for Steering Motors (Steering Motors set to zero).

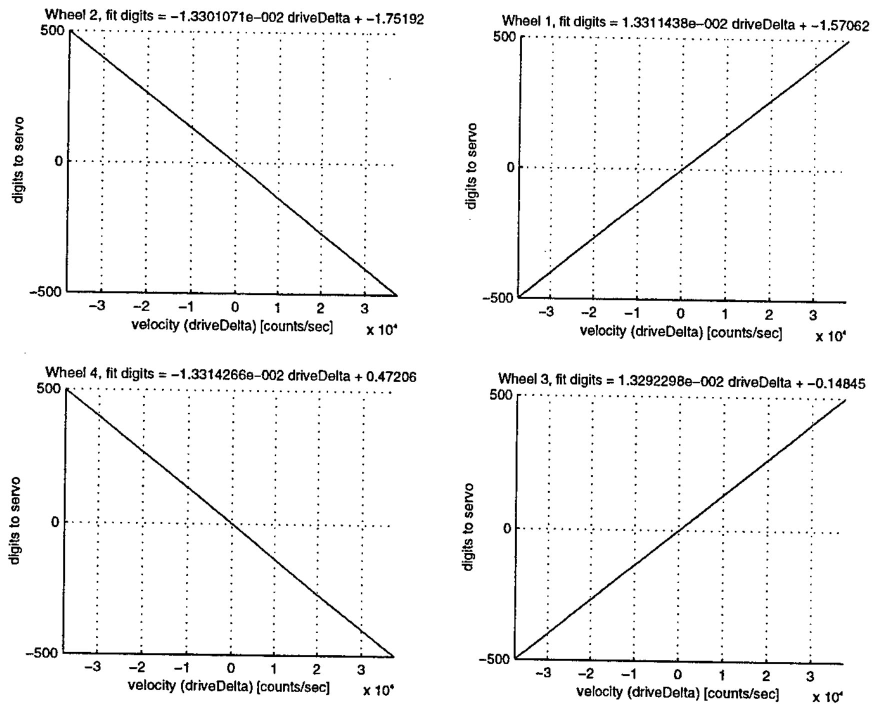


Figure 4.3: Commanded Digits versus Encoder Differences for Vehicle on the Ground.

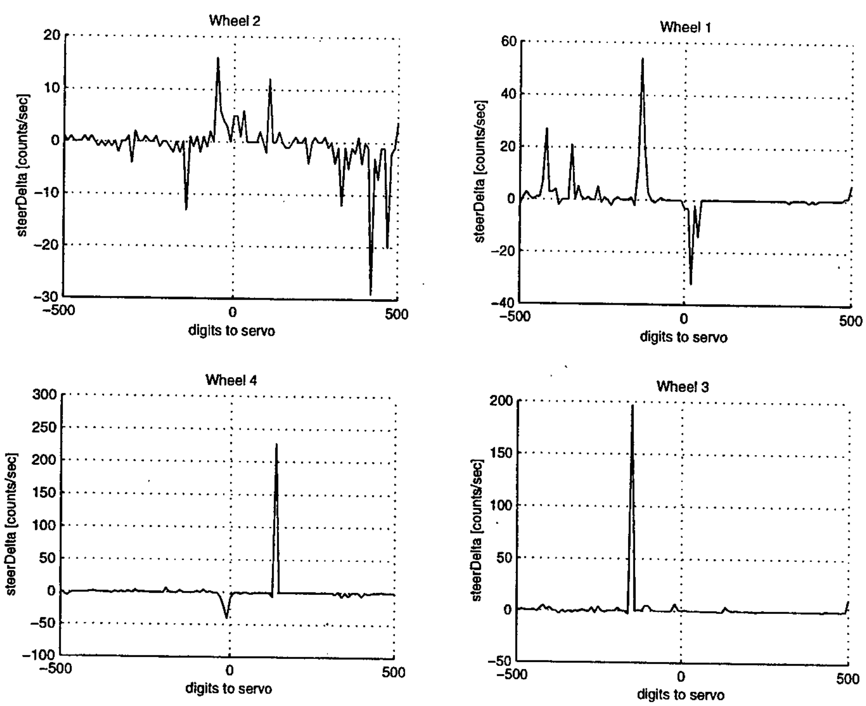


Figure 4.4: Influence of Commanded Drive Digits on Steering Wheels for Vehicle on the Ground. Plot shows Encoder Differences vs. Commanded Drive Digits for Steering Motors (Steering Motors set to zero).

line and a third time with the vehicle lifted off the ground and the wheels rotating free. Despite the changing test conditions, the results were independent from the way the vehicle was suspended. The recorded data for each wheel was fitted in a least square sense by a polynomial of order 1 (a straight line) and the coefficients are given in Table 4.5 where the encoder difference `driveDelta` is given in units of counts per second.

Wheel 1	digit = 0.01331 driveDelta [count/sec] - 1.65
Wheel 2	digit = -0.01330 driveDelta [count/sec] - 1.65
Wheel 3	digit = 0.01329 driveDelta [count/sec] - 0.30
Wheel 4	digit = -0.01331 driveDelta [count/sec] + 0.55

Table 4.5: Relationship between drive encoder difference and commanded servo drive speeds.

It is beneficial to use the relationship digit=f(driveDelta/sec) vice the inverse since for any motion control process, we are given the desired speed (which is directly proportional to the variable driveDelta/sec) and want to obtain the required digit to control the servos accordingly. Using the conversion factor given for driving the wheels (see Table 4.4) and the wheel's radius (which we assume to be equal for all wheels to be 18.9cm) we obtain the conversion from distance travelled to count advances by

$$\begin{aligned} 1 \text{ count} &= \frac{2\pi}{360 * 290} \cdot 18.9 \text{ cm} = 1.13747 \cdot 10^{-3} \text{ cm} \\ 1 \text{ m} &= 87914 \text{ counts} \end{aligned} \tag{4.1}$$

and we finally end up with a handy relationship between velocity [cm/sec] and digits commanded to the servos (the digits are not yet left justified):

Wheel 1	digit = 11.70 v [cm/sec] - 1.65
Wheel 2	digit = 11.69 v [cm/sec] - 1.65
Wheel 3	digit = 11.68 v [cm/sec] - 0.30
Wheel 4	digit = 11.70 v [cm/sec] + 0.55

Table 4.6: Relationship between Velocity [cm/sec] and Commanded Servo Digit (needs further be multiplied by 16 to justify left).

After multiplying the above data by 16 in order to shift it digital wise one nibble to the left, we obtain

Table 4.7 yields the values that can be directly sent to the driving servos. They will already yield the left-justified data sent to the analog output board. Recall that only the upper 12 bit determine the final servo speed. Hence, when driving the wheels, we encounter a discretization error introduced by converting the double valued velocity to 12 bit!

B. LINEAR MOTION PROFILE

In order to test the sampling results obtained from both, the shaft encoder and the IMU, a simple linear motion profile was implemented in the SRK. The profile is implemented as routine `'linearMotion1()'` in the source file `'motor.c'` and is shown in Figure 4.5. As it turned out later, this profile was not suitable to obtain reliable data. Hence, a second profile was implemented as routine `'linearMotion2()'` and the vehicle's principle behavior is depicted in Figure 4.6. While the vehicle would travel a distance of 4 m in forward direction and return to its start position upon execution of `'linearMotion1()'`, it would travel for 5/6 of a meter forward and stop for `'linearMotion2()'`. However, the vehicles maximum acceleration for the former motion would be 2 cm/sec^2 while for the latter, the vehicle would speed up to 1 m/sec^2 which is quite high!

In the following, the results for the shaft encoders for both motion profiles will be discussed utilizing the motion control procedure as outlined in Chapter II on page 12. The analyzing MATLAB routine `'shaft.m'` is for completeness given in Appendix B.5 on page 65.

1. Linear Motion Profile #1

This motion segment lasts for a total of 70 seconds, after which the vehicle is expected to have returned to its start position. The stop during the period $30sec < t < 40sec$ is utilized to mark the turning position for the vehicle.

Clearly, as Figure 4.8 reveals, the driving angles are off by up to 10 degrees upon completion of the motion program. On the floor, a lateral deviation of approximately 35 cm has been observed. The longitudinal distances traveled came out to be 395 cm for the forward leg and 401 cm for the reverse leg.

Despite the fact that the steering motors are set to zero, there remains interaction between driving and steering. It needs to be determined whether or not this relates to badly adjusted (offset) servo motors or indeed driving interaction. In any case, it is quite evident that feedback is required to provide the desired accuracy for straight motion. The aspects of feedback are not discussed in

Wheel 1	digit = 187.20 v [cm/sec] - 26.4
Wheel 2	digit = 187.04 v [cm/sec] - 26.4
Wheel 3	digit = 186.88 v [cm/sec] - 4.8
Wheel 4	digit = 187.20 v [cm/sec] + 8.8

Table 4.7: Relationship between Velocity [cm/sec] and Commanded Servo Digit.

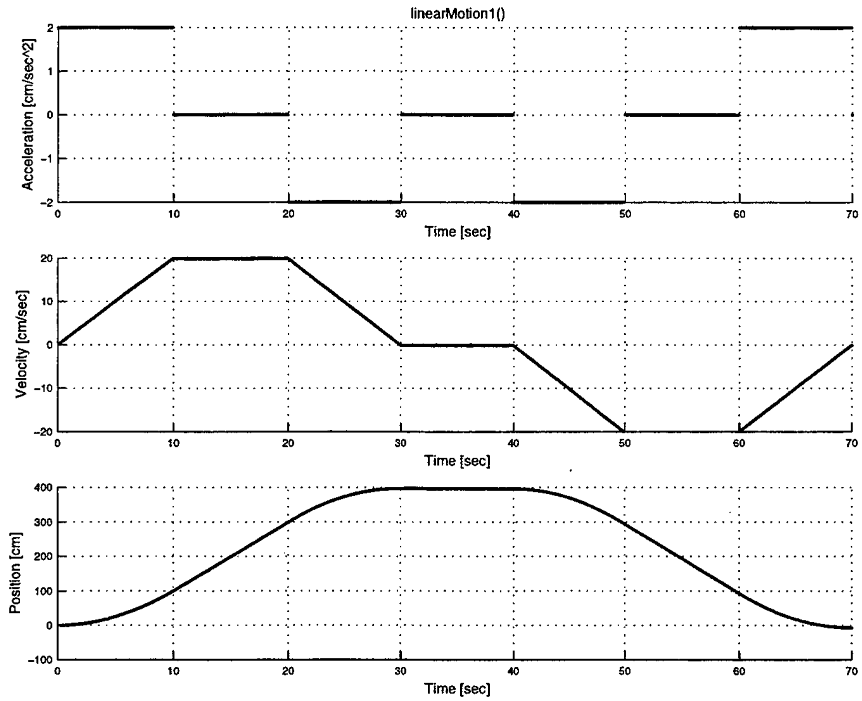


Figure 4.5: Linear motion profile implemented as `linearMotion1()`.

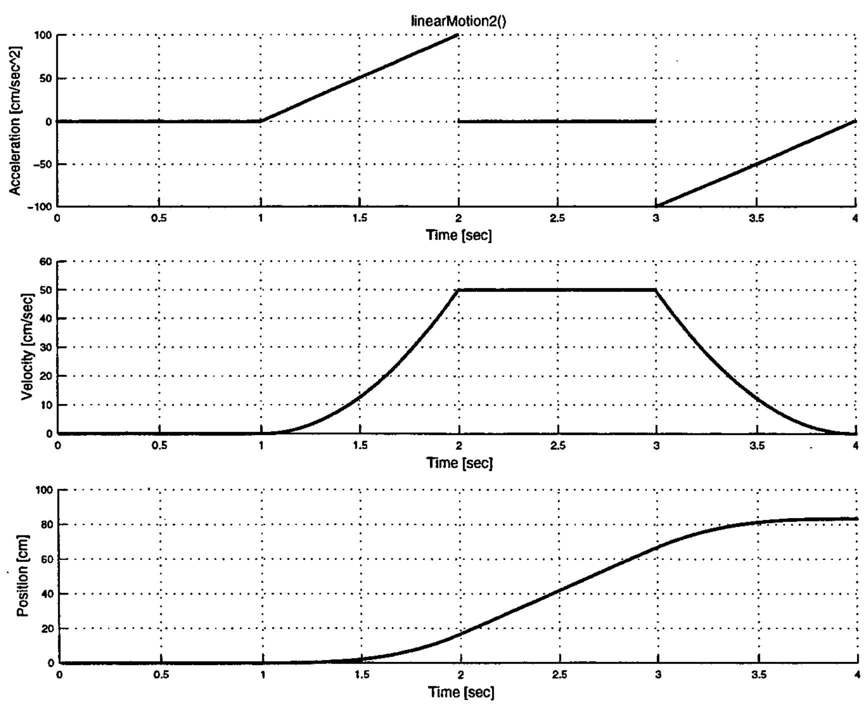


Figure 4.6: Linear motion profile implemented as `linearMotion2()`.

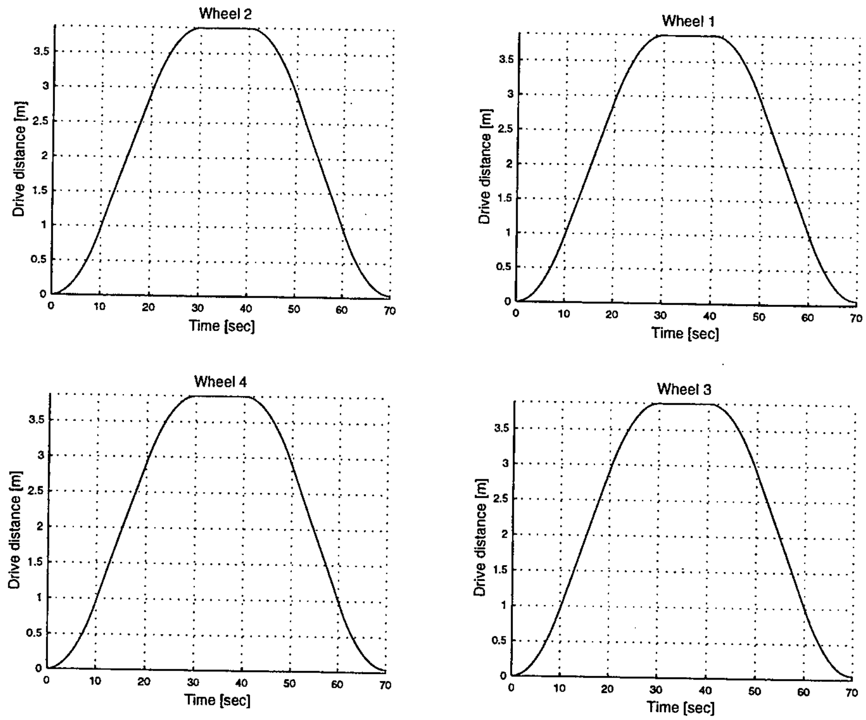


Figure 4.7: Accumulated drive encoder readings versus time for linear motion profile #1.

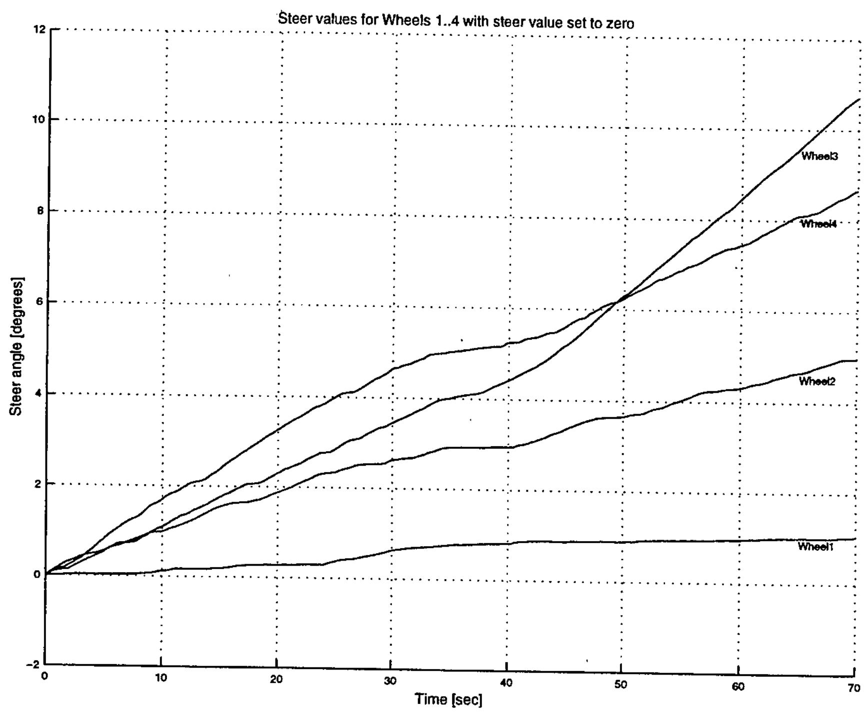


Figure 4.8: Accumulated steer encoder readings versus time for linear motion profile #1.

this thesis. However, Mays/Reid [1] provide a brief discussion about this topic.

2. Linear Motion Profile #2

In order to serve the IMU analysis better, a linear motion profile was needed which provided a greater acceleration for the vegicle. Thus, the linear motion program `'linearMotion2()'` has been implemented in the file `'motor.c'`. This motion program drives the vehicle over a distance of about 83 cm (5/6 m) within 4 sec. As was for the motion profile #1, the vehicle follows closely the determined path.

Considering the fact that no feedback has been implemented in the motion control programs, it can be concluded that the shaft encoder readings provide sufficient accuracy for determining the planar motion for SHEPHERD under the condition that no slip occurs.

C. UNCERTAINTIES IN MOTION CONTROL

It is quite obvious that the accuracy of the motion control part and the position determination depends on several parameters that may vary over time or that were determined too inaccurate. The main reasons for inaccurate motion control and position determination derived from the shaft encoder readings are

1. Inaccurate sensor parameters relating to the angular position of each motor.
2. Wheel radius not measured correctly or radius changing over time due to wear or changing tire pressure.
3. Data reduction for velocity from double valued data type to 12 bit that are being sent to the servos.

All these factors will eventually degrade the performance of the implemented routines. Hence, there will be ample space for improvement for future work.

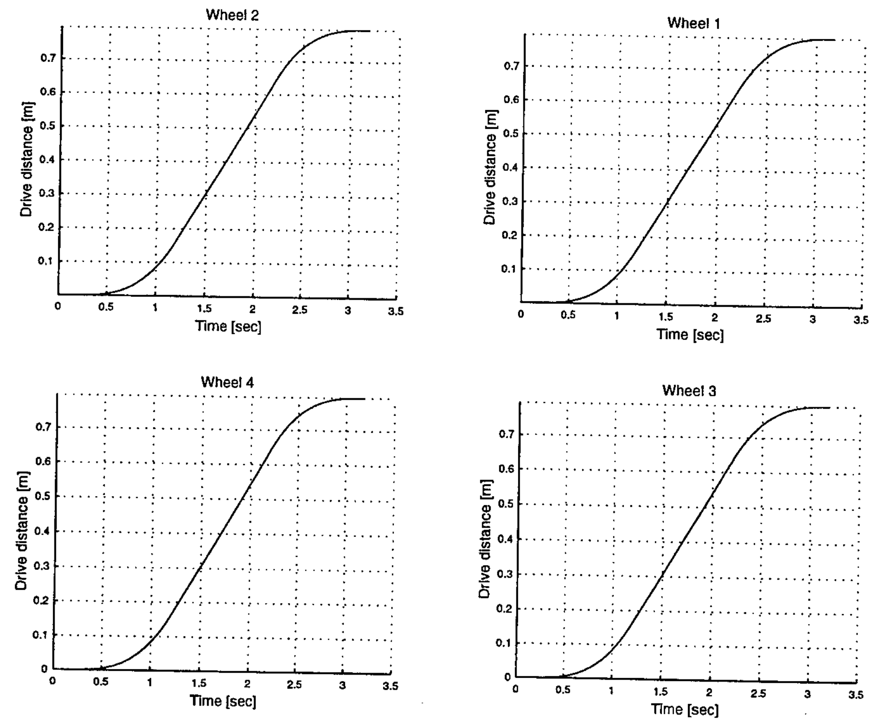


Figure 4.9: Compounded drive encoder readings versus time for linear motion profile #2.

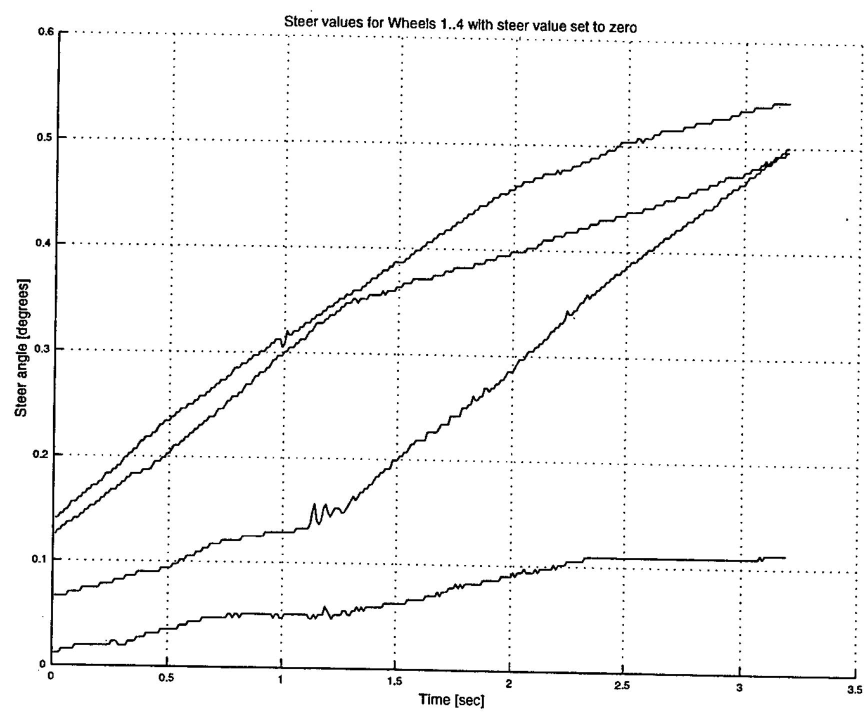


Figure 4.10: Compounded steer encoder readings versus time for linear motion profile #2.

V. INERTIAL MEASUREMENT UNIT

This chapter describes the framework that was implemented on SHEPHERD in an attempt to obtain reliable velocity and position data based on inertial measurements. All source code as it pertains to the implementation of the Inertial Measurement Unit (IMU) is provided in the source file 'imu.c' and listed in Appendix C.1 starting at page 67.

Figure 5.1 shows the vehicle's basic appearance with the four wheels at the corners labelled 1 to 4 and the motion sensor with its three corners marked by a solid dot which span the xy-plane in the body frame {B} mounted on its steel plate. The solid dots on the sensor's casing are just to relate the upside down orientation to the general appearance as given by Figure 5.2.

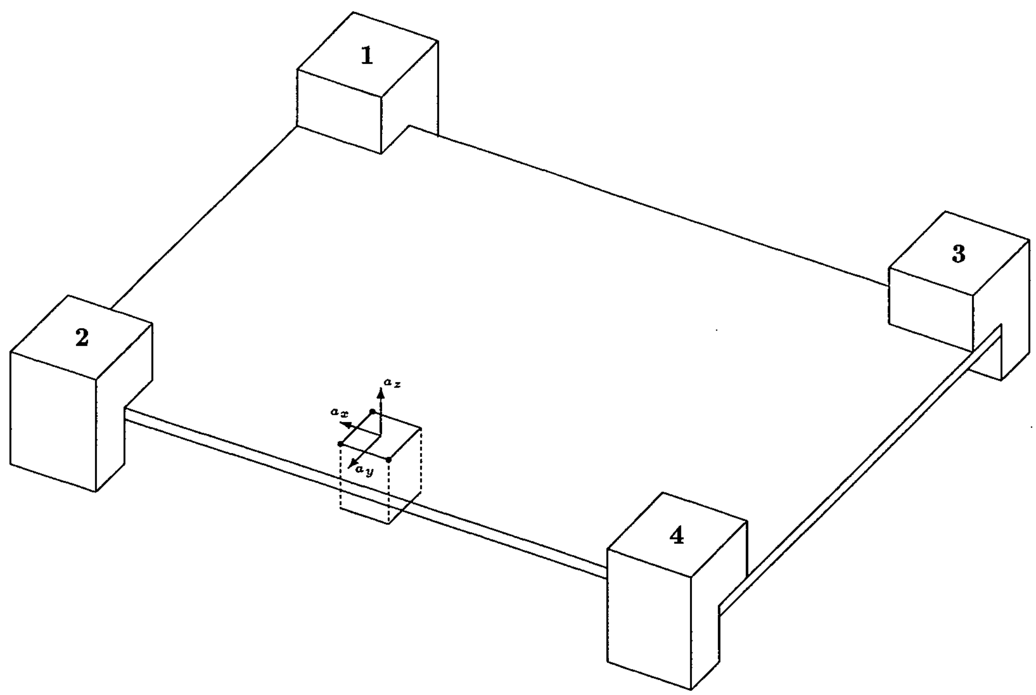


Figure 5.1: Configuration for Shepherd Rotary Vehicle

Due to the particular design of the SHEPHERD Rotary Vehicle, the vertical axes of each wheel are exactly located on the corners of a square of dimension 0.8×0.8 m. The sensor is mounted upside down below the supporting steel plate at the location indicated in Figure 5.1.

A. INERTIAL SENSOR

For this project, a four degree of freedom inertial sensor cluster (Solid-State Motion Sensor, Type MotionPak) from SYSTRON Donner, Concord California [8] is being used. It provides three outputs for linear motion measured with servo accelerators (a_x, a_y, a_y) and one output for measuring rotational motion about the z-axis (ω_z). This data comprises a cartesian coordinate system which is shown in Figure 5.2. The dots in the three corners shall help identify the attitude of the sensor as shown in Figure 5.1.

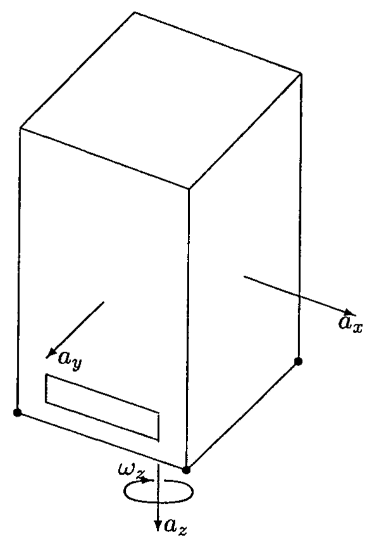


Figure 5.2: Axis orientation for MotionPak Sensor

The MotionPak is customized by the manufacturer for the anticipated dynamic range. Table 5.1 shows most of the specifications as they apply to the model in use.

	x-axis	y-axis	z-axis	
	a_x	a_y	a_z	ω_z
Range	$\pm 2g$	$\pm 2g$	$\pm 2g$	$\pm 50°/sec$
Scale factor	$3.748V/g$	$3.752V/g$	$3.744V/g$	$49.881mV/(deg/sec)$
Stationary output	0.0 V	0.0 V	+3.75 V	0 V
Bandwidth	869 Hz	925 Hz	869 Hz	75 Hz
Noise (10-100Hz)	1.8 mV_{RMS}	1.8 mV_{RMS}	2.0 mV_{RMS}	3.9 mV_{RMS}

Table 5.1: Operating specifications for MotionPak Model No. MP-G-CQBBB-100, Serial No. 0329 (after Reference [9])

As was already shown by Figure 2.4 on page 8, the analog data provided by the MotionPak IMU is converted into digital data by an A/D-Board interfacing to the VMEBus. The converted

digital data is transferred from the A/D-Board to the 68040 processor on the TUARUS board via the VMEBus. Figure 5.3 shows how the four analog channels from the MotionPak IMU are actually routed through the A/D-Board to the CPU.

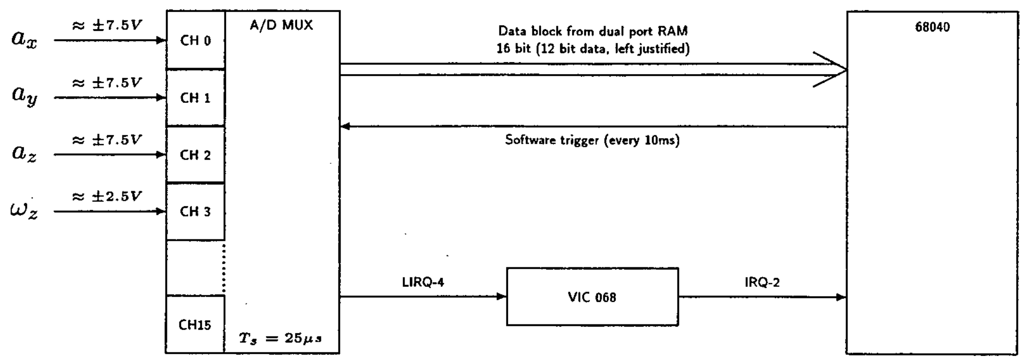


Figure 5.3: IMU Hardware Integration

B. A/D CONVERSION SCHEME

The IMU provides continuous analog data to channels 1 to 4 of the A/D-Board VME9325 [10]. With every 10 ms timer interrupt, a block conversion on the AD-Board is triggered via software command issued by the interrupt handling routine from the 10 ms timer. The AD-Board is configured to multiplex the four input channels every 50 μ sec for a total of 200 samples. Thus, in a consecutive order, each of the four channels are sampled at a sampling rate of f_s=5000 Hz and the digital data is stored sequentially in the A/D-Boards dual-port RAM. Once the block conversion is complete, the A/D-Board will issue an interrupt (see Appendix D.4 on page 93 for the exact interrupt level in use) to 68040 where the corresponding interrupt handler routine `analyzeVME9325()` preprocesses (filters) the block data and stores it as the most recent data in the global variables

$$
\begin{aligned}
a_x &\Longrightarrow \mathsf{imuAX} \\
a_y &\Longrightarrow \mathsf{imuAY} \\
a_z &\Longrightarrow \mathsf{imuAZ} \\
\omega_z &\Longrightarrow \mathsf{imuOmegaZ}
\end{aligned}
$$

which will thus be available for the next motion control cycle to update the actual vehicle motion. The board's status can be observed by means of LED indicator lights at the boards front panel:

Green LED	Red LED	Status
off	on	Board is not initialized
on	on	Board undergoes initialization
off	off	Board is initialized but inactive
on	off	Board is performing A/D block conversions

Table 5.2: Status indicator lights for A/D-Board

At present, the data is merely downloaded via the TAURUSBug 'du0' option (see Appendix D.3) through the CONSOLE port to the Laptop and from there to the UNIX System, where the data was further analyzed using MATLAB. However, for the future, the sampled data would be directly processed by the 68040 processor as outlined above.

One might ask, why was the odd sampling frequency f_s = 5000 Hz is being used instead of a more intuitive 10 kHz. A look at the timing diagram Figure 5.4, reveals that the time Δ between the last block conversion (ω_z in block 50) and the start of the next motion control cycle is governed by the sampling frequency: for continuous sampling (e.g., increased block number to transfer), the larger f_s the smaller will Δ be. However, there is a constraint on the minimum length of Δ due to the fact that the sampling block data must be transferred to the TAURUS main memory. This transfer must be done before the next motion control cylce is issued by the 10 ms timer interrupt. This rule must be closely followed, otherwise a loss of sampling data might occur.

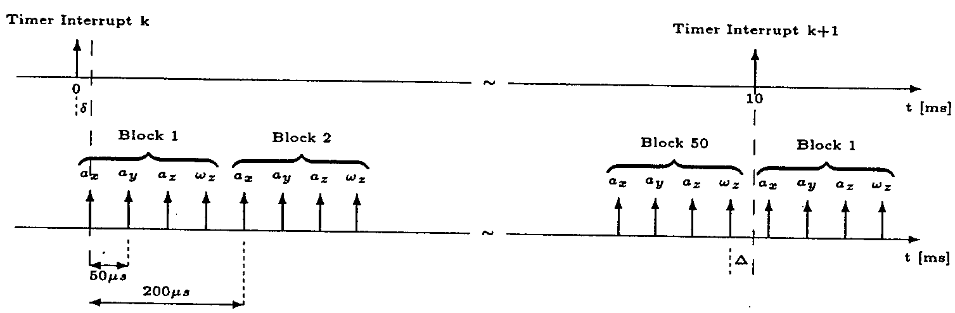


Figure 5.4: Timing Diagram for A/D-Board

The A/D-Board maps a preset input span of Δ = 20 V for a differential input range of $\pm$ 10 V into n=12 bit bipolar two's complement data left justified in a 16 bit word. The value of -2048 relates to an analog input equvalent of -10 V $\leq x_{analog} <$ -9.99512 V. Likewise, the digital output

of 2048 relates to 0 V $\leq x_{analog} < 0.00488$ V. The stepsize is given by $\delta = \frac{\Delta}{2^n} = \frac{20V}{4096} = 4.88$ mV. To make use of the maximum range available, the board provides a variable gain to amplify the input signal by factors G=1, G=2, G=4, or G=8. Moreover, we need to scale the data by the appropriate scaling factors S for each channel which are given in Table 5.1. Thus, for a given channel with gain G and scaling S, we obtain the analog equivalent of the data by shifting the digital value $x_{digital}$ by 4 bit to the right (which is equivalent to a division by 16) and than re-scale it according to:

$$x_{analog} = \frac{\Delta}{2^n\, G\, S} \left(x_{digital} - 2048\right) \quad .$$

Using the scaling factors given in Table 5.1 we end up with the units of [g] for a_x, a_y, and a_z and [degrees/sec] for ω_z. Expressing the linear acceleration a in terms of the gravitational acceleration g rather than in SI-units of $[m/sec^2]$ turns out to be beneficial if we need to find the Euler angles and a suitable representation for it in the reference frame {R}.

C. SCHEME FOR DATA ANALYSIS

Accelerometers sense the sum of the gravitational acceleration $\vec{a}_g$ and the linear acceleration $\vec{a}$ which is due to an external force applied to the body in the body frame {B}

$$^{\mathrm{B}}\vec{\mathrm{a}}_m = {}^{\mathrm{B}}\vec{\mathrm{a}} + {}^{\mathrm{B}}\vec{\mathrm{g}} \tag{5.1}$$

which relates to the reference frame {R} as

$$^{\mathrm{R}}\vec{\mathrm{a}}_m = {}^{\mathrm{R}}\vec{\mathrm{a}} + {}^{\mathrm{R}}\vec{\mathrm{g}} \quad . \tag{5.2}$$

In both frames, g is the acceleration of gravity derived from Keplerian physics for two body motion theory between the Earth and a body. Usually, g is a function of the distance r between the center of masses of the two bodies and can be computed with

$$g = \frac{G\, M}{r^2}$$

with the constants G and M as described in Appendix A. For a body at the Earth's surface, $g \approx 9.81\ m/sec^2$ and usually, the variation in height for small changes can be neglected. Therefore we will not concern ourselves with a variable g and assume that $g = 9.81\ m/sec^2$.

In the following, we will devise a scheme to eliminate the undesired gravity components in our measurement data. Therefore, we will have to focus on the stationary vehicle first, that is, the only acceleration acting on the vehicle in frame {B} will be the Earth gravity. Moreover, we know that in the reference frame {R}, the acceleration due to gravity has only a +z-component whereas

in {B} we would usually encounter gravitational components in each of the principal axes unless the sensor is perfectly aligned with frame {R}:

$$
{}^{\mathrm{R}}\vec{\mathrm{g}} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \qquad \text{and} \qquad {}^{\mathrm{B}}\vec{\mathrm{g}} = \begin{pmatrix} {}^{\mathrm{B}}\mathrm{g}_x \\ {}^{\mathrm{B}}\mathrm{g}_y \\ {}^{\mathrm{B}}\mathrm{g}_z \end{pmatrix}
$$

subject to the constraint that $g = \sqrt{{}^{\mathrm{B}}\mathrm{g}_x^2 + {}^{\mathrm{B}}\mathrm{g}_y^2 + {}^{\mathrm{B}}\mathrm{g}_z^2}$. To express frame {B} in terms of frame {R} we make use of the rotation matrix as outlined in the previous sections and given by Equation 3.2:

$$
{}^{\mathrm{R}}\vec{\mathrm{a}}_m = {}^{\mathrm{R}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{B}}\vec{\mathrm{a}}_m
$$

We therefore do need to get the Euler Angles (roll, pitch, and yaw) as defined on page 17. We make us of the fact that the acceleration of a stationary sensor as measured in {R} should only display the gravitation:

$$
{}^{\mathrm{R}}\vec{\mathrm{a}}_m = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Z}}(\psi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Y}}(\phi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{X}}(\theta)\,{}^{\mathrm{B}}\vec{\mathrm{a}}_m \qquad .
$$

Solving for ${}^{\mathrm{B}}\vec{\mathrm{a}}_m$ yields

$$
{}^{\mathrm{B}}\vec{\mathrm{a}}_m = {}^{\mathrm{R}}\mathrm{R}_{\mathrm{X}}^{-1}(\psi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{y}}^{-1}(\phi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Z}}^{-1}(\theta)\ {}^{\mathrm{R}}\vec{\mathrm{a}}_m \qquad .
$$

We recall the identity given in Equation 3.1 on page 16 and rewrite the above equation in terms of the transpose of each rotation matrix:

$$
{}^{\mathrm{B}}\vec{\mathrm{a}}_m = {}^{\mathrm{R}}\mathrm{R}_{\mathrm{X}}^{T}(\psi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{y}}^{T}(\phi)\ {}^{\mathrm{R}}\mathrm{R}_{\mathrm{Z}}^{T}(\theta)\ {}^{\mathrm{R}}\vec{\mathrm{a}}_m \qquad . \tag{5.3}
$$

For any measurement vector ${}^{\mathrm{B}}\vec{\mathrm{a}}_m$ and the related vector ${}^{\mathrm{R}}\vec{\mathrm{a}}$ in frame {R}, Equation 5.3 together with the definitions for the rotation matrices Equation 3.3, Equation 3.4 and Equation 3.5 given on page 17 provides us with a system of three equations from which we can determine the Euler Angles. In particular, we are easily able to determine the Euler angles as a function of the measurement ${}^{\mathrm{B}}\vec{\mathrm{a}}_m$:

$$
\begin{aligned}
a_{xm} &= -g\ sin(\phi) && (5.4)\\
a_{ym} &= g\ sin(\theta)\ cos(\phi) && (5.5)\\
a_{zm} &= g\ cos(\theta)\ cos(\phi) \quad . && (5.6)
\end{aligned}
$$

We recognize that for the stationary data, the acceleration measured in {B} does not depend on the yaw angle ψ which is directly related to the heading of the vehicle (in order to obtain the heading, we,

of course, would need to have a compass at hand). Solving the above system for the two remaining Euler angles yields the following equations:

$$\phi = -arcsin\left(\frac{a_x}{g}\right) \tag{5.7}$$

$$\theta = arcsin\left(\frac{a_y}{g\ cos(\phi)}\right) \tag{5.8}$$

or alternatively for θ

$$\theta = arcsin\left(\frac{a_y}{\sqrt{a_y^2 + a_z^2}}\right) \quad . \tag{5.9}$$

We see that the last two equations both yield a solution for θ. Depending on the accuracy of our measurements and the accuracy of the desired math functions we have implemented so far, we may prefer the one to the other. Since the Sensor's output data is already scaled with respect to g, the Earth's gravity (see Table 5.1), we may prefer the former and discard Equation 5.9. This is reflected in the MATLAB listing for 'getdata.m' where the data is arranged accordingly.

Based on the theory pertaining to the inertial measurement sensor as outlined above, the following scheme to obtain the position data for the vehicle is proposed:

1. Sample stationary data (as is usually the case if one starts up the vehicle) in frame {B} for a certain period of time.
2. Filter the data with an appropriate lowpass filter.
3. Compute the Euler angles θ and ϕ.
4. Transform the data from frame {B} to frame {R} using the rotation matrices given by Equation 3.2, use arbitrary yaw angle ψ.
5. Subtract the acceleration due to gravity acting on the vehicle to obtain the sole acceleration due to a specific force given in frame {R}.
6. Integrate the data in a suitable way to find the velocity and position vector of the vehicle.

D. INTEGRATION TOOLS

In our analysis of the inertial measurement sensor, we will have to integrate the data in order to arrive at the velocity vector. There are many integration methods available for integrating discrete data. For equispaced, discrete data, most of the more commonly known integration formulas such as the Trapezoidal rule, Simpson's Rule, ... are based on the Newton-Côtes Integration Formulas ([11],[12]). Given a set of values $f(x_i)$ for equispaced $x_i = a + ih\ \forall\ i = 0 \ldots n$ with $h = \frac{b-a}{n}$, the

integral of $f(x)$ on the interval $[a, b]$ can be approximated by

$$\int_a^b f(x)\, dx = \int_a^b P_n(x)\, dx$$

where $P_n(x)$ is the Lagrangian polynomial that passes through all the points x_i and the interval $[a, b]$ is covered by the (n+1) equidistant points x_i. $P_n(x)$ is given by

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\, \alpha_i$$

where α_i is given by

$$\alpha_i = \prod_{\substack{k=0 \\ k \neq i}}^{n} \left(\frac{x - x_k}{x_i - x_k} \right) \quad .$$

If we let $x = a + hs$ the above integral for $P_n(x)$ reduces to a simple sum

$$\int_a^b P_n(x)\, dx = h \sum_{i=0}^{n} f(x_i)\, \alpha_i = \frac{b-a}{ns} \sum_{i=0}^{n} \sigma_i f(x_i) \quad . \tag{5.10}$$

The values for ns and σ_i can be computed given the above relations. However, we will not concern ourselves with this issue and state the results for the first few parameters:

n	ns	σ_i	Commonly known rule
1	2	1 1	Tapezoidal
2	6	1 4 1	Simpson's 1/3
3	8	1 3 3 1	Simpson's 3/8
4	90	7 32 12 32 7	
5	288	19 75 50 50 75 19	
6	840	41 216 27 272 27 216 41	

Table 5.3: Newton-Côtes Formula Parameters

Some of these formulas are being implemented in the function '`integral.m`' on page page 65 and used for integrating the acceleration data. The analysis in the following sections will discuss which formula shall be preferred to the others.

E. DATA FILTERING AND COMPUTATION OF POSITION VECTOR

Several recordings for stationary data have been taken. In the process of obtaining the position vector for the vehicle we would expect that starting, say from an initial position $(0, 0, 0)_{\{R\}}$, this should not vary much as time passes by.

Initially, the sampling scheme was such that each channel of the IMU was sampled at a sampling rate of 100 Hz with every 10 ms timer interval. Later on, this has been changed to a sampling rate of 5000 Hz as shown in the timing diagram Figure 5.4 on page 34.

1. Stationary Data Analysis

The data collected for the stationary data analysis in this subsection has been sampled prior to changing the sampling frequency from 100 Hz to 5000 Hz. Thus, this is reflected in the data presented in this subsection. In addition, the IMU at this stage was not yet mounted to the vehicle and the orientation of the axes was such that the sensors z-axis pointed up instead of down as shown in Figure 5.1. Figures 5.5 to 5.10 show typical results obtained. They show data recorded and processed for a stationary vehicle with file 'imu.m' (see Appendix B.1 on page 59). The data was recorded on the fifth floor of Spanagel Hall with the sensor titled by a significant amount which was not further specified.

As can be seen from Figure 5.6, the linear components (a_x, a_y, and a_z) contain distinct sinusoidal components at $f = 20Hz$ and $f = 40Hz$. The origin of this behavior still needs further examination. However, it seems not to be related to the block sampling interval of T=10 ms, rather than to vibrations inherent in the building. These sinusoidal components can not be beneficial to the performance of our compuations. Therefore, we have to eliminate the residues by some suitable filtering technique.

In the time domain (Figure 5.5), we see the effect due to the A/D sampling process: the sampled data obtained through the A/D Board truly displays the characteristics for discrete-time signals. Moreover, since the sensor was titled, the data will reflect the values according to this orientation relative to frame {R}. Thus, the next step involves computation of the Euler angles and transforming the data into frame {R} using the results obtained in Equation 3.2. Now, follwoing the transformation the data for a_x and a_y should ideally go to zero (at least in the mean). The result is shown in Figure 5.7 with its Fourier spectrum given by Figure 5.8.

In fact, the acceleration for a_x and a_y is almost zero whereas the acceleration for a_z is almost -1.0 g (the DC component is not shown in the frequency spectrum. The negative sign for this data set is due to the fact that the sensor's z-axis pointed down. The final step is to obtain the velocity and the position by integrating the acceleration once or twice, respectively. The velocity is shown in Figure 5.9. As can be seen from the plot, the velocity in x- and z-direction pretty much approaches steady-state after about 3 sec of recording whereas the velocity in y-direction approaches steady state after about 10 seconds (eventually, a longer recording needs to be taken to verify this statement). As for the position vector, which is shown in Figure 5.10, we see that during the first second the error is small and the position remains pretty much zero. However, as the velocity assumes its steady state, the position displays a linear behavior. Therefore, based on the stationary analysis, it is advisable to update (reset) the navigation solution based on the IMU at least every second. Even better, if

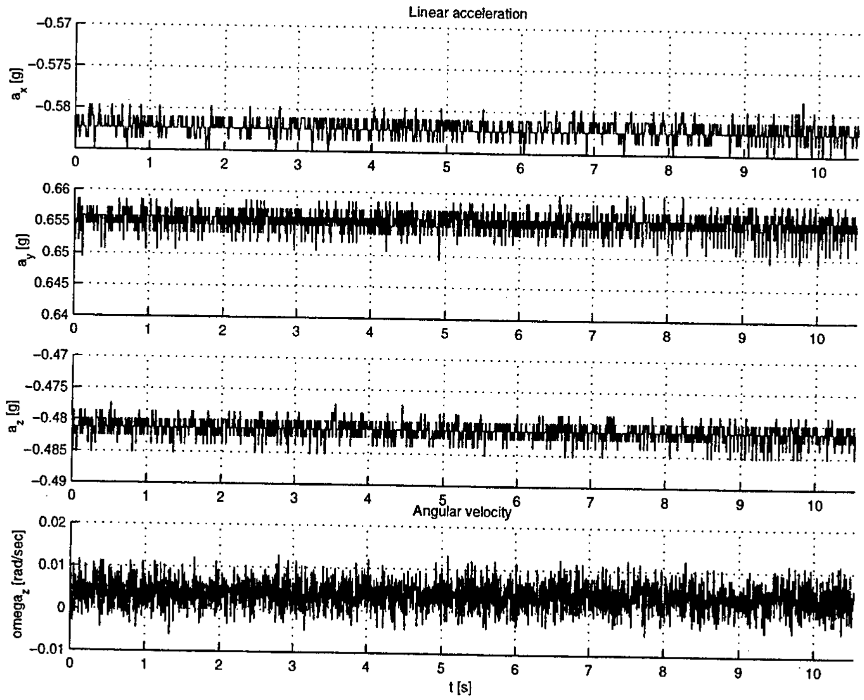


Figure 5.5: Time domain behavior for linear acceleration and angular velocity for the stationary and tilted IMU as measured by the A/D-Board (normalized to units [g]) in frame {S}.

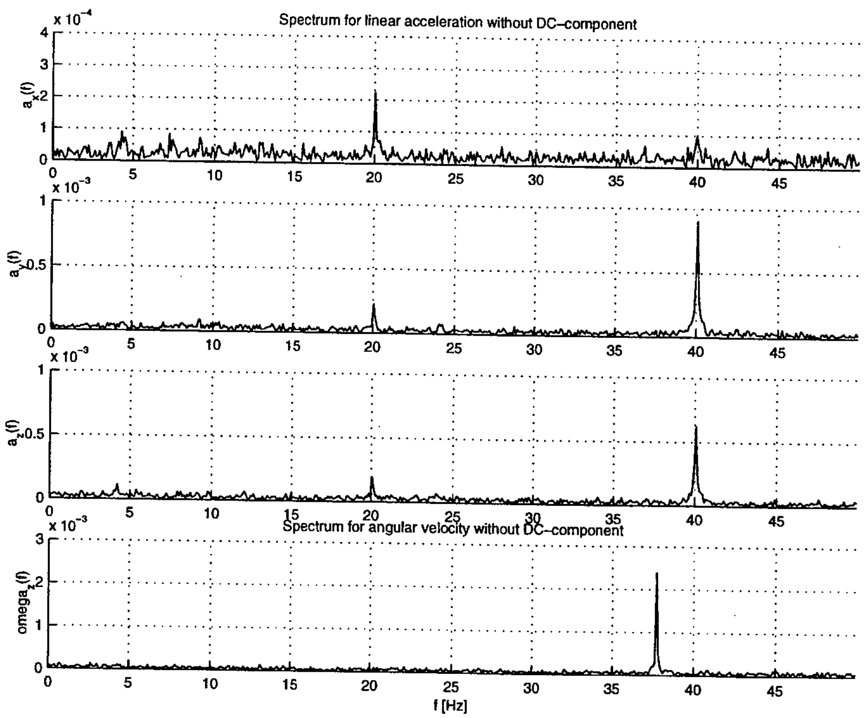


Figure 5.6: Fourier spectrum for linear acceleration and angular velocity for the stationary and tilted IMU as measured by the A/D-Board (normalized to units [g]) in frame {S}.

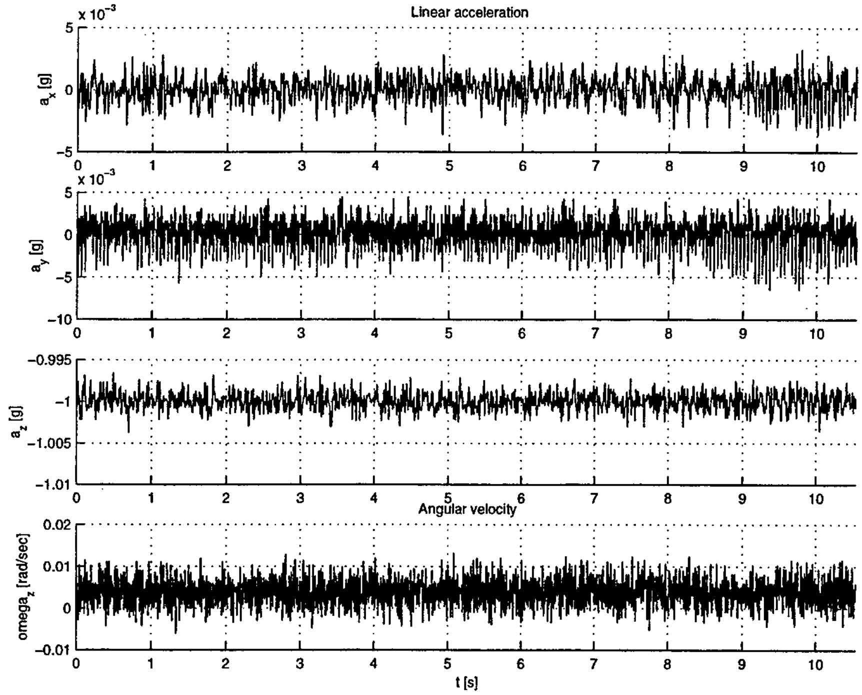


Figure 5.7: Time domain behavior for linear acceleration and angular velocity for the stationary and tilted IMU as measured by the A/D-Board (normalized to units [g]) in the reference frame {R}.

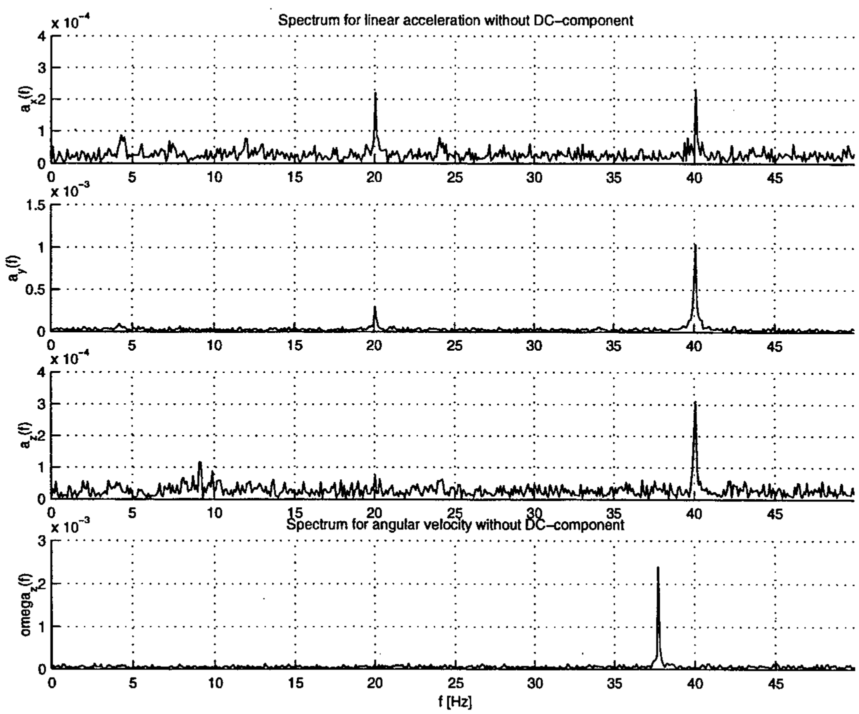


Figure 5.8: Fourier spectrum for linear acceleration and angular velocity for the stationary and tilted IMU as measured by the A/D-Board (normalized to units [g]) in the reference frame {R}.

the Euler angles which represent the attitude of the vehicle could be determined continuously and in accordance to the updated Euler angles, new rotation matrices would have to be determined on a regular basis.

2. Non-stationary Data Analysis with Profile #1

In the sequel, we will analyze data sampled at a sampling frequency of $f_s = 5000$ kHz according to the timing diagram depicted in Figure 5.4 from an IMU that is mounted on SHEPHERD as shown in Figure 5.1. First, in order to correlate the sampled data to the actual motion of the sensor/vehicle, the same linear test motion profile as introduced in Chapter IV and shown in Figure 4.5 on page 27 was being utilized. Due to the vast amount of data that had to be analyzed (a recording for 70 sec at a sampling frequency of 5000 Hz on four IMU channels comprised a mere 2.8 MByte!) the analysis was performed on segments of data in order not to exploit the limits of computational power. In particular, to enhance the performance of the built in MATLAB Fourier transform function, segments contained 65536 samples, which is a power of two (2^{16}).

Figure 5.11 depicts the linear acceleration as determined by the IMU. Despite the fact that the linear motion profile was only along the x-axis of the vehicle, the sensor seemed not to distinguish between the channels. All three components display some sort of noise and the signals do not at all seem to be related to the actual motion profile.

The detailed analysis of the a_x-channel is given in Figure 5.12 and 5.13 for the time frame $0 < t < 13 sec$. Figure 5.12 shows that the original data is distorted throughout the entire frequency range. Moreover, the time signal does not display the expected behavior according to the true motion profile. Instead, the oscillations increase in amplitude as time advances. To reduce the noise, an elliptic filter has been used to attenuate the noise in the stopband. The software filter, implemented using MATLAB's built in signal processing functions, had the following specifications:

1. Passband from $0 \ldots 20$ Hz with max. attenuation of 0.1 dB
2. Stopband from 50... Hz with min. attenuation of 80 dB

Other filters such as Chebychev and Butterworth filters were also being tested. None of these filter types showed a significant improvement of the data. The only advantage Butterworth or Chebychev filters have compared to Elliptic filters is a better phase linearity in the passband. On the other hand, and most important for an implementation where computation time is scarce, Elliptic filters are most efficient since they yield the smallest-order filter for a given set of specifications [14].

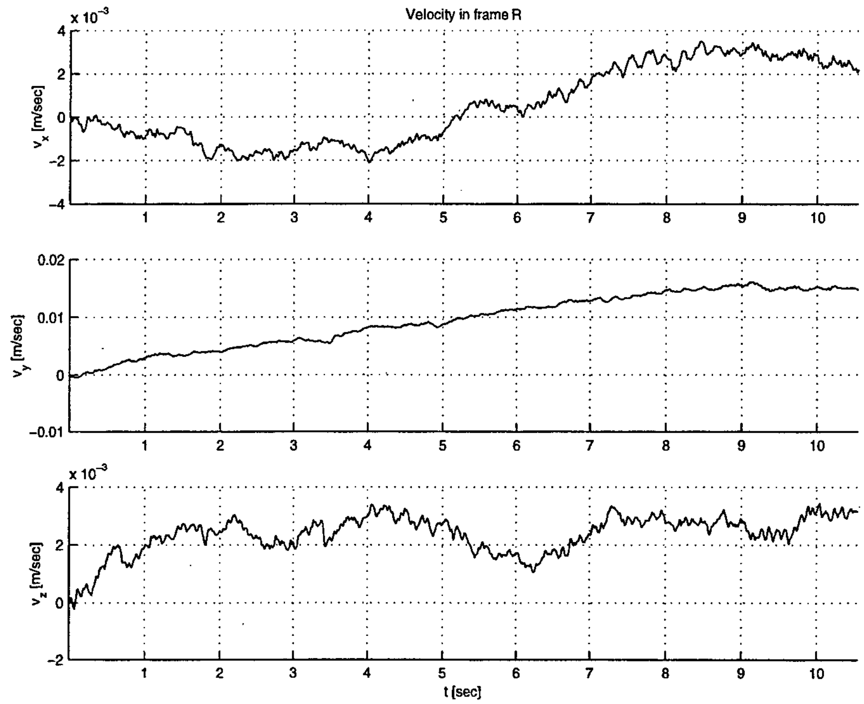


Figure 5.9: Velocity data integrated from the linear acceleration in frame {R}.

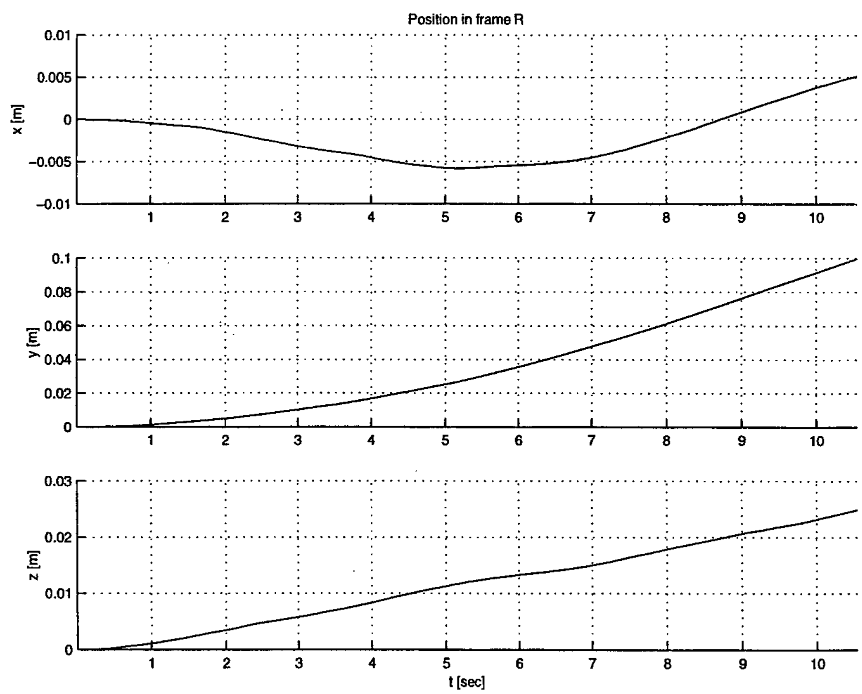


Figure 5.10: Position integrated from the velocity in frame {R}.

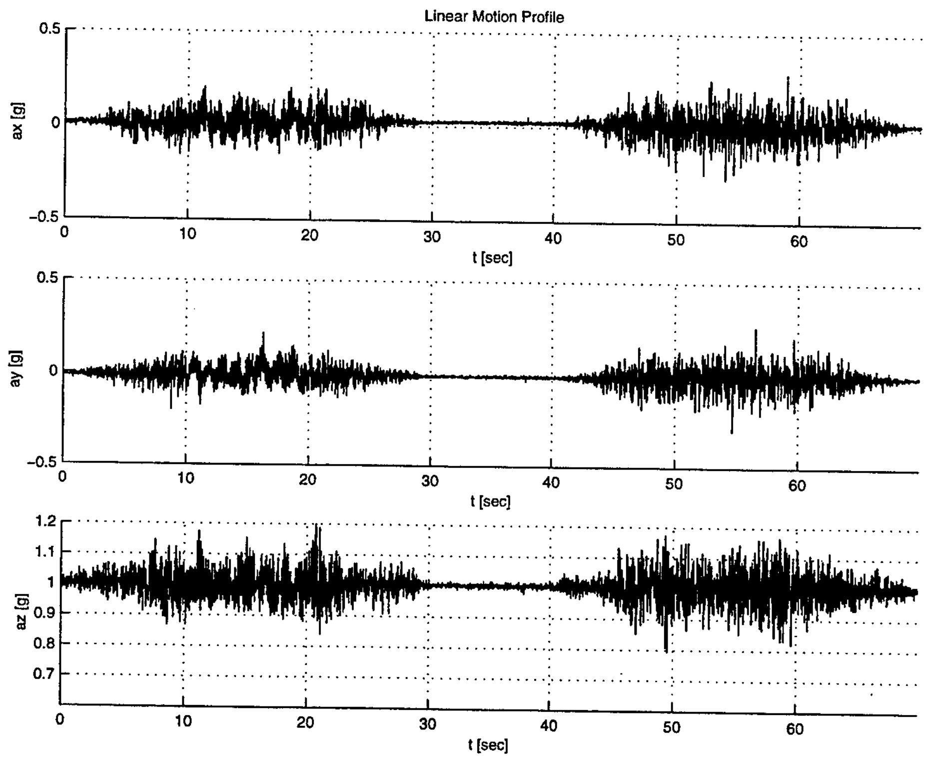


Figure 5.11: Linear Acceleration measured by all three channels of the IMU for Linear Motion Profile #1.

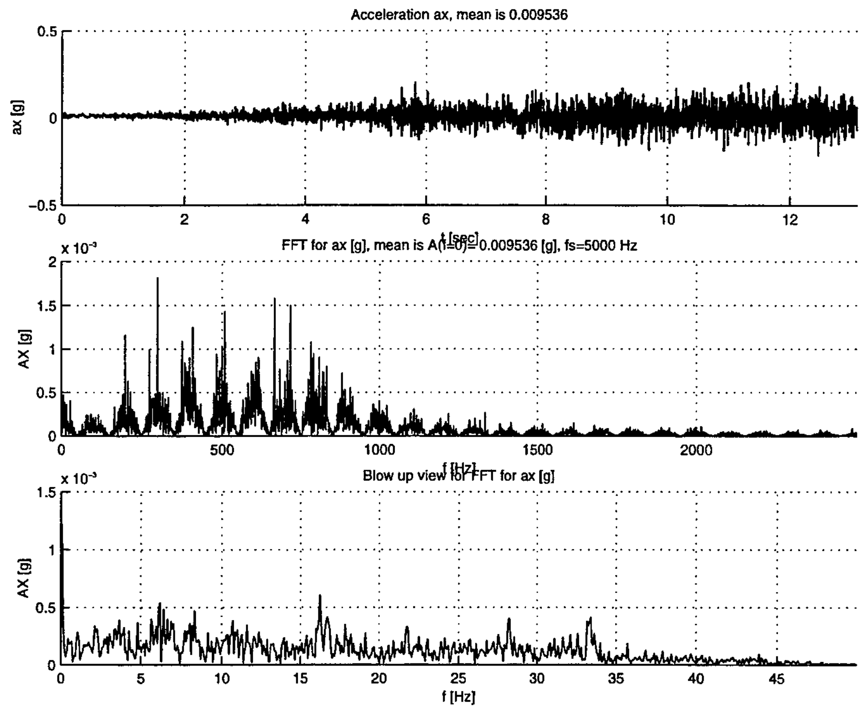


Figure 5.12: Analysis of linear acceleration ax as measured by the IMU.

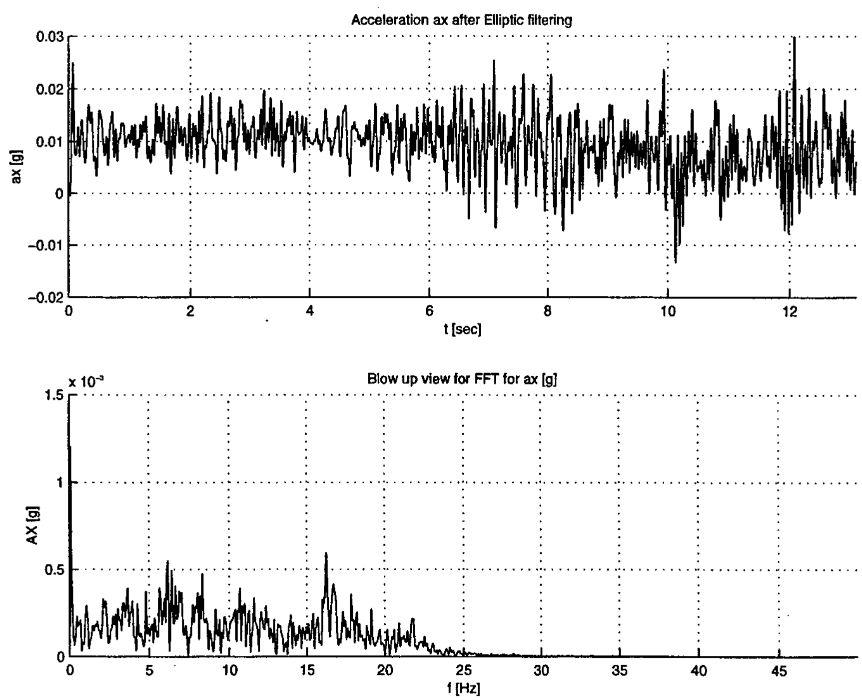


Figure 5.13: Analysis of linear acceleration ax after data has been filtered by a 6th order elliptic filter with passband edge at 20 Hz and Stopband edge at 50 Hz.

The results, as depicted in Figure 5.13 do not look too promising. Althought the filter achieved to smooth the data and reduce the noise, it could not ensure that the acceleration would show any transition at t=10sec. Recall that according to the true profile, the acceleration should be zero starting with t=10sec. The only reason that can be attributed to this fatal behavior is the dynamic input range of the A/D-Board: operating the accelerometer at a maximum linear acceleration of $a_x = 0.02m/sec^2$ (which is only $\approx$ 0.002 g) we utilize only a voltage span from -7.6 mV to +7.6 mV that is fed into the A/D-Board. Even if the maximum gain of 8 is used to amplify this signal, the amplitude would never exceed $\approx$ 62 mV which comprises a mere four digits in the digital output range.

3. Non-stationary Data Analysis with Profile #2

It was anticipated that, for the second motion profile as shown in Figure 4.6, results for the measured acceleration would improve. The maximum acceleration was set to be 1.0 m/sec^2 with the maximum velocity reached by the vehicle to be $\approx$ 0.5 m/s. The sampled data for all three linear acceleration channels is shown in Figure 5.14. The plot reveals strong interaction between all three channels. One goal would be to get rid of these interferences by means of a suitable filter technique. For the time being, we focus on the a_x-channel. The time and frequency behavior for the x-channel is depicted in Figure 5.15. Strong harmonic components influence the overall performance and a similarity to the actual motion can not be found.

Upon filtering with an elliptic filter of order 6, the recorded data can somewhat be related to the true motion. However, since the sharp edges in the ideal acceleration profile (Figure 4.6) result in high frequency components of the signal, these edges can not be recognized by the IMU (the cutoff frequency for the linear accelerometers is around 900 Hz, see Table 5.1. Nonetheless, the questions remains: would this be suffice to compute the velocity? We refer to Figure 5.16 and see that the velocity does in principle follow the curve depicted by the ideal motion profile Figure 4.6. As soon as the recognizable motion kicks in, the velocity seems to be distorted by an offset in the acceleration data (rather than assuming a=0 on the interval $t \in [2,3]$ sec).

4. Non-stationary Data Analysis with Profile #3

To get rid of the lowpass constraint, a third motion profile has been developed. The profile is shown in Figure 5.17.

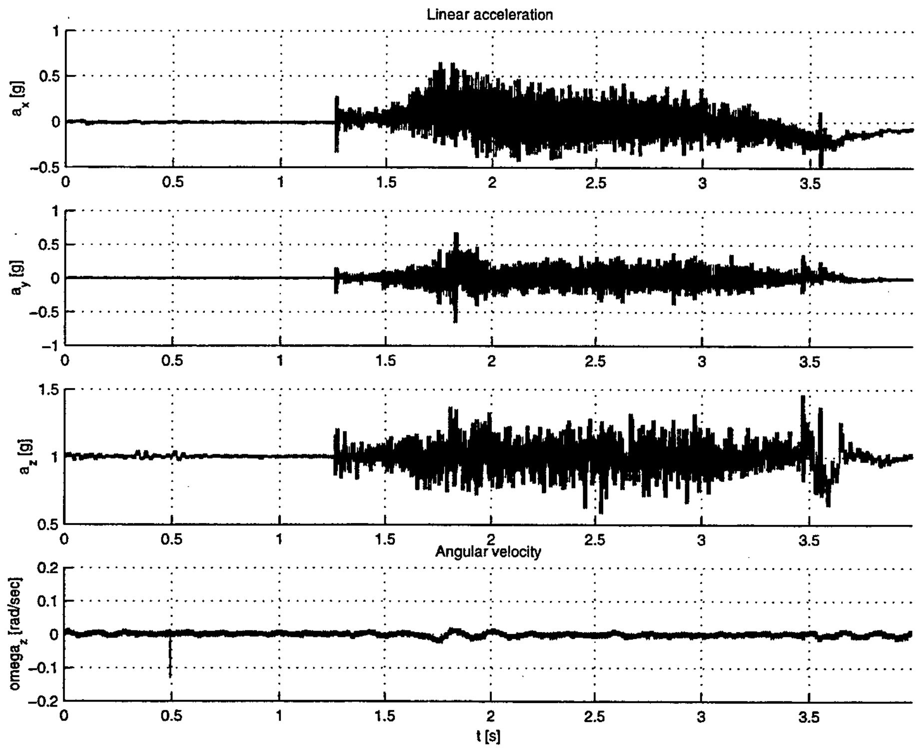


Figure 5.14: Linear Acceleration and angular velocity ω_z relative to frame {R} measured by the IMU for Linear Motion Profile #2.

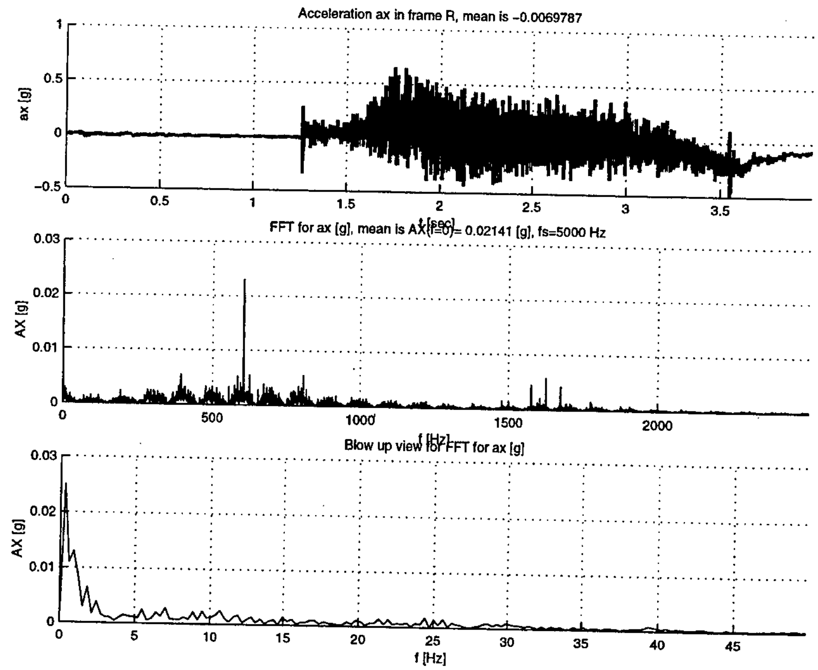


Figure 5.15: Analysis of linear acceleration ax as measured by the IMU.

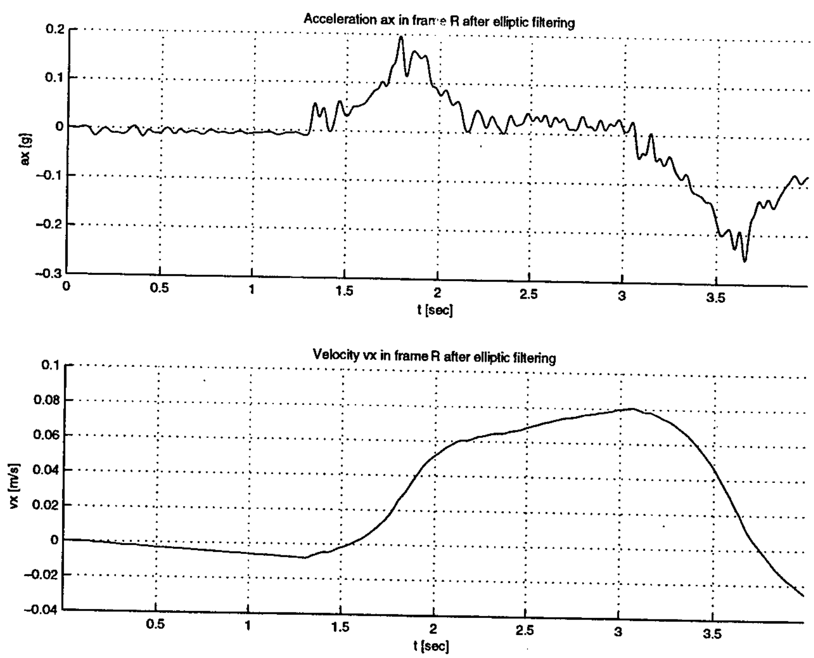


Figure 5.16: Analysis of linear acceleration ax after data has been filtered by a 6th order elliptic filter with passband edge at 20 Hz and Stopband edge at 50 Hz.

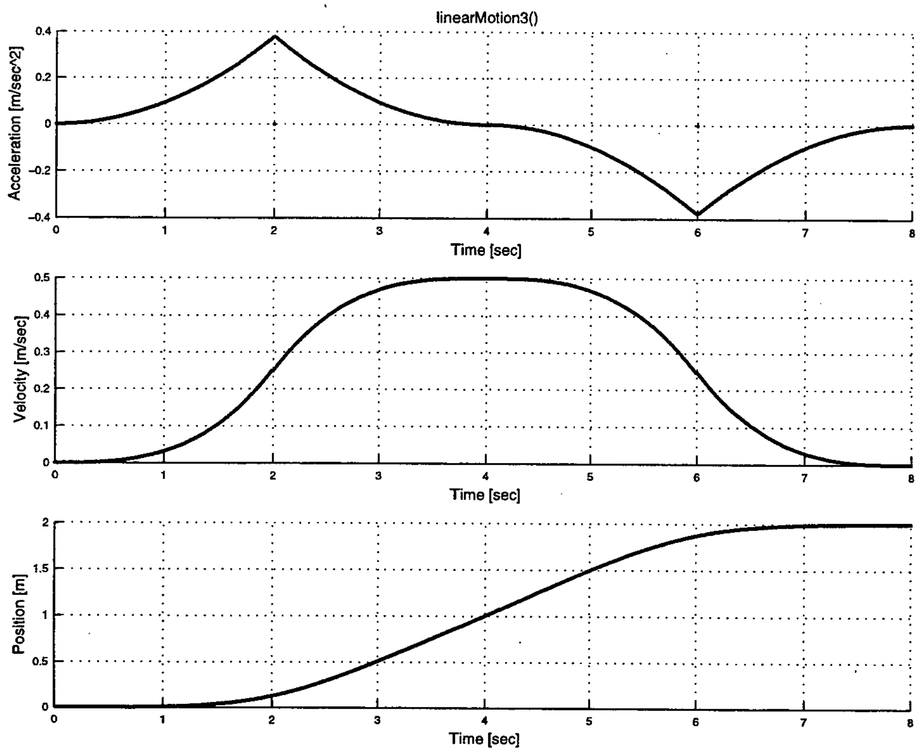


Figure 5.17: Linear Motion Profile #3.

Clearly, this motion should only contain low frequency components. As was the case for the other two motion profiles, the IMU senses noise in all three channels even though the motion takes place only in the sensors x-direction.

F. SUMMARY

Based on the results obtained from the linear motion profiles #1 .. #3 the following conclusions for the implementation of the inertial measurement unit can be drawn: First, the IMU data sampled off the IMU needs to fit appropriately in the A/D-Boards input range. As a crude rule of thumb based on the observations made in this Chapter, the time average of the acceleration signals to be A/D-converted (this may include any additional gain) should be at least 1/10 th of the max. allowable input amplitude of the A/D-Board (e.g., at present, the max. input is $\pm$ 10 V, the input signal should be at least 1 V in magnitude). A more detailed analysis is required in this respect. Probably the most effective solution would be to utilize MotionPak Accelerometers (QFA7000) with current output rather than voltage output. In this case, the output could be scaled by the user to especially lower 'g' limits by means of variable scaling resistors (see [13] for more information). Probably the most significant shortfall in the design of the vehicle was determined to be the variable suspension of the vehicle's wheels. Whenever the vehicle accelerates by a significant amount, the vehicle's steel platform may tilt. This change of attitude will be recognized by the IMU but can not be attributed to a change of the vehicle's main body attitude and thus to a change of position in 3D space.

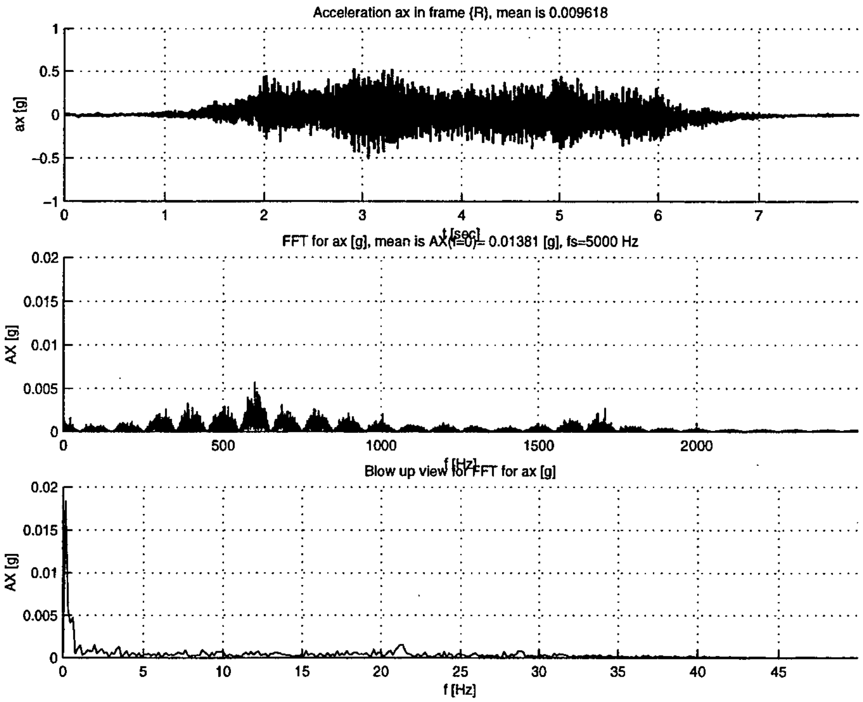


Figure 5.18: Analysis of linear acceleration ax as measured by the IMU.

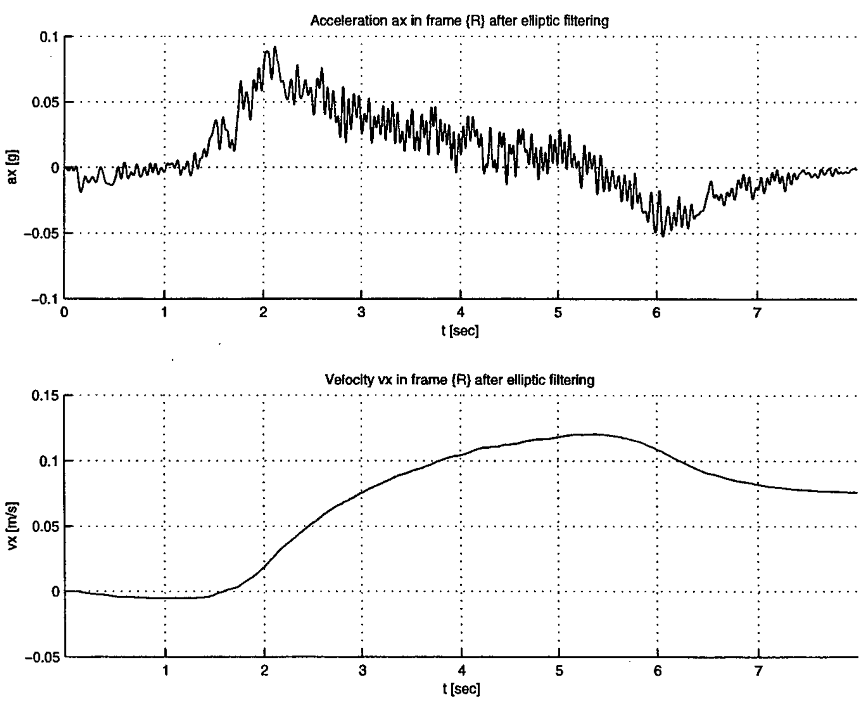


Figure 5.19: Analysis after Elliptic Filtering (6th order filter) with passband edge at 20 Hz and Stopband edge at 50 Hz.

VI. SENSOR FUSION

Having developed the two independent navigation components in the previous Chapters, it was anticipated to fuse the data provided by both systems to further improve the accuracy of the navigation system. However, since the performance of the IMU does not yield any reliable motion data, sensors fusion at this point of time is obsolete. Some literature research has been done to obtain a hint as to how to fuse the data. Almost all papers related to sensor fusion utilize the extended Kalman filter. Welch [16] provides a decent introduction in Kalman filtering. Nonetheless, it is anticipated that Neural Networks might be applicable to this problem as well. Thus, the aspect of sensor fusion will be left for future work.

VII. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

A. CONCLUSIONS

The research issues addressed by this thesis were

- Implement the hardware and software for an Inertial Measurement Unit
- Implement the software for a shaft encoder system
- Evaluate the performance for both sensors
- Sensor Fusion

Both the IMU and the shaft encoder systems have been implemented in software and hardware. The sampling frequency for the A/D-Board was set to be 5 kHz. Both systems have been tested with three different linear motion profiles.

The work conducted in addressing the first of these topics revealed several sources of navigation inaccuracy. The A/D Converter board currently in use does not match the IMU's output range for accelerations below about 1 m/sec^2. In addition, due to the vehicle's sophisticated wheel suspension, the IMU's attitude control could not be related to the attitude of the vehicle and was changing with time as the vehicle moved. This introduced a slowly varying and yet significant error in numerically integrating the acceleration.

The second issue addressed proved to be less difficult. Decent results have been obtained for th elinear motion under the condition that no slip occurs and the vehicle's position can be determined to within 0.5 percent accuracy.

The overall motion control system seems to be stable at all. However, it has been observed that computation power for the 68040 processor is scarce. This is mainly to the fact that a public domain GCC Compiler is in use for generating the executable code. This compiler does not seem to generate optimal executable code. In addition, the lack of a math processor and math library functions required that semi-optimal trigonometric functions be implemented in the source code as well, introducing further inaccuracies.

B. RECOMMENDATIONS FOR FUTURE WORK

There are many issues that were briefly addressed in this thesis but could not be investigated in detail. Much work needs to be done in the following areas.

1. Determine the optimal resolution for the A/D-Board based on the anticipated motion profiles.

2. Investigate whether or not variable gain control for the IMU data would improve the performance of the IMU.

3. Develop a scheme for attitude control vice changing the vehicle's suspension.

4. Implement the filter algorithms as determined in this thesis. Care needs to be taken that computation time is crucial and efficient computation methods be used.

5. Implement an Input/Output Kernel utilizing the 68030 processor for online debugging, display of status information, and eventually off-loading of some of the lower priority task such as transferring data between boards.

6. Investigate how the system presented in this thesis would work for most general type of motion including rotational motion and motion in three dimensions.

APPENDIX A: CONSTANTS

Table 1.1: Constants used throughout the text

Universal constant of gravitation	G=$6.672 \cdot 10^{-11}\ \frac{m^3}{kg\ sec^2}$
Mass of Earth	M=$5.98 \cdot 10^{24}\ kg$
mean Earth radius	R_e=$6.371 \cdot 10^{6}\ m$

APPENDIX B: MATLAB M-FILES

This appendix contains essential MATLAB M-Files that are being referenced in the text.

1. IMU.M

The MATLAAB file 'imu.m' is used to analyze the data recorded from the IMU. It makes use of the MATLAB functions 'filter1', 'euler1.m' and 'integral' which are listed following this section.

```
function imu(fname,G,T,f)

% ---------------------------------------------------------------
% function imu(fname,G,T,f)
% ---------------------------------------------------------------
%
% M-File to obtain reliable position data. Procedure:
%
%   1. load data and scale data
%   2. plot data in frame {B}
%   3. filter data with butterworth LP filter in frame {B}
%   4. determine Euler angles and transform data fto frame {R}
%   5. integrate data to obtain velocity
%
% Author:     Thorsten Leonardy
% Date:       10/23/97
% Compiler:   MATLAB V4.21c
%
% Input:      fname = name of data file
%             G     = gain sequence for channels, default [1 1 1 4]
%                     note that G(3) includes the orientation of the
%                     IMU's z-axis (>0 is up, <0 is down)
%             T     = sampling time for data
%             f     = switch for filtering ax data
% ---------------------------------------------------------------

g=9.81;               % local gravitational constant [g=9.81m/s^2]

if nargin<2
   G=[1 1 1 4];       % sample gain
   T=0.01;            % samples per block and channel
   f=0;               % do not filter data
end

up = G(3)/abs(G(3))   % determine if IMU's z-axis points up
G(3)=abs(G(3));

% load data, ax,ay and az are in [m/sec^2] or [g], wz is in [rad/sec]
[t,ax,ay,az,wz]=getdata(fname,G,T);

disp('>>> Plot data in {B} ...')
plotdata(t,ax,ay,az,wz);                % plot data

disp('>>> Transform {B} --> {R} ...')
[ax,ay,az]=euler1(ax,ay,az,up);         % transform data to reference frame {A}

disp('>>> Plot data in {R} ...')
plotdata(t,ax,ay,az,wz);                % plot data in {R}

disp('>>> Integrate data in {R} to obtain v ...')
[tv,vx]=integral(t,g*ax,1);             % integrate step by step
[tv,vy]=integral(t,g*ay,1);             % integrate step by step
[tv,vz]=integral(t,g*(az-up),1);        % integrate step by step
```

```

figure
myplot(tv,vx,'Velocity in frame {R}','','v_x [m/sec]',[3 1 1])
myplot(tv,vy,'','','v_y [m/sec]',[3 1 2])
myplot(tv,vz,'','t [sec]','v_z [m/sec]',[3 1 3])

disp('>>> Integrate data in {R} to obtain position ...')
[tp,x]=integral(tv,vx,1);               % integrate step by step
[tp,y]=integral(tv,vy,1);               % integrate step by step
[tp,z]=integral(tv,vz,1);               % integrate step by step

figure
myplot(tp,x,'Position in frame {R}','','x [m]',[3 1 1])
myplot(tp,y,'','','y [m]',[3 1 2])
myplot(tp,z,'','t [sec]','z [m]',[3 1 3])

% ----------------------------------------------
% filter the data for acceleration in x direction
% ----------------------------------------------
if f
  mx=mean(ax);                        % compute the mean
  my=mean(ay);                        % compute the mean
  mz=mean(az);                        % compute the mean


  % compute the FFT
  [AX,f]=filter1(ax,6,t(2)-t(1));

  mAX=AX(1);                          % obtain the mean
  AX(1)=0;                            % suppress dc component

  figure
  myplot(t,ax,['Acceleration ax in frame {R}, mean is ' num2str(mx)],'t [sec]','ax [g]',[3 1 1])

  myplot(f,AX,['FFT for ax [g], mean is AX(f=0)= ' num2str(mAX) ' [g], fs=5000 Hz'],...
           'f [Hz]','AX [g]',[3 1 2])

  % zoom on in for f=0..50 Hz
  ix=find(f<=50);
  myplot(f(ix),AX(ix),'Blow up view for FFT for ax [g]',...
              'f [Hz]','AX [g]',[3 1 3])

  % --------------------------------------------------
  % filter the data
  % --------------------------------------------------
  af=filter1(ax,10,20/2500,50/2500,0.1,80);  % Cauer filter

  figure
  myplot(t,af,'Acceleration ax in frame {R} after elliptic filtering',...
              't [sec]','ax [g]',[2 1 1])



  % --------------------------------------------------
  % Integrate ax
  % --------------------------------------------------
  [t,v]=integral(t,af,6);

  myplot(t,v,'Velocity vx in frame {R} after elliptic filtering',...
              't [sec]','vx [m/s]',[2 1 2])

end % of if f

disp('>>> Plot all figures to disk in postscript format as ''fname_xxx.ps''')
for i=1:gcf
  figure(i)
  eval(['print -dps2 ' fname '_' num2str(i) '.ps'])
end

return
% -----------------------------------------------------------------
% end of 'imu.m'
% -----------------------------------------------------------------
```

2. FILTER1.M

The file `'filter1'` provides a set of suitable filter routines such as an FFT, Chebychev or Butterworth filter, and more.

```
function [y,f]=filter1(x,type,a,b,c,d)
% ----------------------------------------------------------------
% function [y,f]=filter1(x,type,a,b,c,d)
% ----------------------------------------------------------------
% Author:     Thorsten Leonardy
% Date:       10/16/97
% Compiler:   MATLAB V4.2c1
%
% Input:      x = input data matrix (M*N)
%             type = utility function (filter) to apply
%             a..d = parameter used for some filter types
%
% type 2..4 average across the rows:
%             type = 2 simple mean
%             type = 3 average using Simpson's 3/8 rule
%             type = 4 average using Simpson's 1/3 rule on 9 samples
%             type = 5 average using trapezoidal rule
% type 6 operate on each row:
%             type =  6 obtain Fourier transform (a is the sample interval in [sec])
% type 7 ... 9 operate on first row only:
%             type =  7 moving average FIR-Filter         [n Taps]
%             type =  8 Butterworth filter           [wp,ws,Rp,Rs]
%             type =  9 Chebychev Filter             [wp,ws,Rp,Rs]
%             type = 10 Elliptic (Cauer Filter)      [wp,ws,Rp,Rs]
%
% Output:     y = output data (M*N2/2),
%                 N2 is a power of two closest to and less or equal to N
%             f = frequency scale (1*N2/2) for y if type=10
% ----------------------------------------------------------------

disp(['*** Function "filter1", type ' num2str(type) ' ***'])

if type==0
   y=x;
   return
end

% compute mean of the sampled data from the channel
if type==1
  y=x(a,:);
end

if type==2
  y=mean(x);
end

if type==3
  c=(3/8)*[1 3 3 2 3 3 2 3 3 1];
  y=c*x/9;
end

if type==4
  c=(1/3)*[1 4 2 4 2 4 2 4 1];
  y=c*x(1:9,:)/8;
end

if type==5
  c=(1/2)*[1 2 2 2 2 2 2 2 2 1];
  y=c*x/9;
end

% ---------------------------------------------------------------------
% Fourier Transform of x
% ---------------------------------------------------------------------

if type==6

   T=a;                            % sampling time of data
```

```
    F=1/T;                          % sampling frequency [Hz] of signal
    m=mean(x');                     % mean of data sequence

    N=size(x,2);                    % total length of data
    N2=2^(floor(log(N)/log(2)))     % reduced length to power of two
    x(:,N2+1:N)=[];                 % cut off the data sequence
    t=T*(0:N2-1);                   % time base corresponding to data
    f=linspace(0,F,N2);             % frequency base

    % Matlab computes the Fourier transform of a signal that is sampled
    % at a sampling frequency fs. The corresponding frequency scale is
    % expressed in terms of the digital frequency omega=2*pi*(f/fs) in
    % the range 0..2*pi (any discrete FT is periodic in terms of omega
    % with period 2*pi).

    y=abs(fft(x'))';          % compute the Fourier Transfor of x(t)
    f(:,N2/2+1:N2)=[];        % discard redundant frequency part
    y(:,N2/2+1:N2)=[];        % discard redundant upper half of spectrum
                              % X(w) relates now to w=[0,pi]
    y=y/N2;                   % normalize the amplitude

end


% ************** moving average FIR filter *********************
if type==7

   if nargin<3
      P=5;
   end
   M=P;
   N=size(x,2);

   x=x(1,:);                             % filter only first row

   x=x-[ zeros(1,1+M) x(1:N-1-M)];       % the delay
   x=x/(1+M);

   y=zeros(1,N);
   y(1)=x(1);

   for i=2:N
      y(i)=y(i-1)+x(i);
   end

end


% ---------------------------------------------------------------------
% IIR Butterworth filter
% ---------------------------------------------------------------------
if type == 8

   x=x(1,:);                             % filter only first row

   % filter specifications (digital frequencies)
   % e.g. if fs=2000Hz and passband edge is supposed to be at fp=500 Hz,
   % parameter wp must be wp=fp/(fs/2)=500/(2000/2)=0.5 !!!
   wp=a;      % wp is passband edge [0..1] where 1 relates to fp/(fs/2) ...
   ws=b;      % stopband edge ...
   Rp=c;      % ... and max. attenuation [dB] at passband edge
   Rs=d;      % ... and min. attenuation [dB] at stopband edge
   [N,wc]=buttord(wp,ws,Rp,Rs);   % filter order and 3dB cutoff-frequency
   disp(['Butterworth filter order ' num2str(N)])

   %filter process
   [b,a]=butter(N,wc);                  % compute the filter coefficients
   y=filter(b,a,x);                     % filter the data

end

% -------------------------------------------------
% Chebychev Type II filter
% -------------------------------------------------
if type==9

```

```
    x=x(1,:);                          % filter only first row

    % filter specifications (digital frequencies)
    % e.g. if fs=2000Hz and passband edge is supposed to be at fp=500 Hz,
    % parameter wp must be wp=fp/(fs/2)=500/(2000/2)=0.5 !!!
    wp=a;      % wp is passband edge [0..1] where 1 relates to fp/(fs/2) ...
    ws=b;      % stopband edge ...
    Rp=c;      % ... and max. attenuation [dB] at passband edge
    Rs=d;      % ... and min. attenuation [dB] at stopband edge

    [N,wn]=cheb2ord(wp,ws,Rp,Rs);   % filter order and 3dB cutoff-frequency
    disp(['Chebychev Type II filter order ' num2str(N)])

    [b,a]=cheby2(N,Rs,wn);             % compute the filter coefficients
    y=filter(b,a,x);                   % filter the data

end

% ---------------------------------------------------
% Elliptic filter (Cauer filter)
% ---------------------------------------------------
if type==10

    x=x(1,:);                          % filter only first row

    % filter specifications (digital frequencies)
    % e.g. if fs=2000Hz and passband edge is supposed to be at fp=500 Hz,
    % parameter wp must be wp=fp/(fs/2)=500/(2000/2)=0.5 !!!
    wp=a;      % wp is passband edge [0..1] where 1 relates to fp/(fs/2) ...
    ws=b;      % stopband edge ...
    Rp=c;      % ... and max. attenuation [dB] at passband edge
    Rs=d;      % ... and min. attenuation [dB] at stopband edge

    [N,Wn]=ellipord(wp,ws,Rp,Rs);   % filter order and 3dB cutoff-frequency
    disp(['Elliptic filter order ' num2str(N)])

    [b,a]=ellip(N,Rp,Rs,Wn);              % compute the filter coefficients
    y=filter(b,a,x);                   % filter the data

end


return

% -------------------------------------------------------------------
% end of 'filter1.m'
% -------------------------------------------------------------------
```

3. EULER1.M

The function `euler1.m` is used to convert the recorded IMU data which is given in the sensor frame {S} to the reference frame {R} by means of rotation matrices.

```
function [ax,ay,az]=euler1(ax,ay,az,up)

% -------------------------------------------------------------------
% function [ax,ay,az]=euler1(ax,ay,az,up)
% -------------------------------------------------------------------
%
% M-File for computing the Euler angles for a given set of data
% measured in the sensor frame {S} and transforming the data into
% the reference frame {R}.
%
% Author:     Thorsten Leonardy
% Date:       10/16/97
% Compiler:   MATLAB V4.21c
%
% Input:      ax(1,N) =  acceleration [g] in {S} ax-direction
%             ay(1,N) =  acceleration [g] in {S} ay-direction
```

```
%               az(1,N) =  acceleration [g] in {S} az-direction
%               up      =  orientation of sensors z-axis (+1=up,-1=down)
%
% Return:       acceleration relative to frame {R}
% ---------------------------------------------------------------

% put data into one measurement matrix aS(3,N) relative to Frame {S}
aS=[ax;ay;az];

% ---------------------------------------------
% determine the Euler angles based on the average
% acceleration during 2nd second
% ---------------------------------------------
ix=101:200;          % may change this
m=mean(aS(:,ix)');   % take the mean of first ix values
g=sqrt(m*m');        % the gravity based on the mean
disp(['--> mean of g in frame {S} is ' num2str(g,6) ' g'])

psi=0.0;                             % psi, arbitrary value
phi=-asin(m(1));                     % phi
theta=asin(m(2)/cos(phi));           % theta

phi=up*phi;
theta=up*theta;

disp(['--> Theta (roll) is ' num2str(theta*180/pi,7) ' degrees'])
disp(['--> Phi (pitch) is ' num2str(phi*180/pi,7) ' degrees'])
disp(['--> Psi (yaw) is ' num2str(psi*180/pi,7) ' degrees'])

% ---------------------------------------------
% compute elements of the rotation matrix
% complete rotation matrix would be R=RZ*RY*RX
% ---------------------------------------------

RX=[    1          0            0       ;     % rotation matrix about X_A
        0      cos(theta)  -sin(theta)  ;
        0      sin(theta)   cos(theta)  ];

RY=[ cos(phi)      0         sin(phi)   ;     % rotation matrix about Y_A
        0          1            0       ;
    -sin(phi)      0         cos(phi)   ];

RZ=[ cos(psi) -sin(psi)         0       ;     % rotation matrix about Z_A
     sin(psi)  cos(psi)         0       ;
        0          0            1       ];

% ---------------------------------------------
% rotate the data successively to frame {A}
% ---------------------------------------------
aR=RX*aS;         % rotate {B} about {R} x-axis
aR=RY*aR;         % rotate new {B} about {R} y-axis
aR=RZ*aR;         % rotate new {B} about {R} z-axis

m=mean(aR(:,ix)');  % take the mean of first ix values
g=sqrt(m*m');       % the gravity based on the mean
disp(['--> mean of g in frame {A} is ' num2str(g,6) ' g'])

ax=aR(1,:);
ay=aR(2,:);
az=aR(3,:);

return
% ---------------------------------------------------------------
% end of 'euler1.m'
% ---------------------------------------------------------------
```

4. INTEGRAL.M

This function implements the Newton-Cotes integration formulas as described in the text. This provides an easy means to compare the results for different integration schemes.

```
function [t,y]=integral(t,x,n)

% ---------------------------------------------------------------
% function [t,y]=integral(t,x,n)
%
% Integrates the input x based on the Newton-Cotes algorithm.
% The integral is computed on each column.
%
% n = the number of panels (n panels require n+1 data points)
% t is the time base corresponding to the data.
% ---------------------------------------------------------------

[N,c]=size(t)

if (c>N)
   x=x'; t=t'; N=c;   % need data as a vector, N=length of data
end

% prepare the coefficients in the sum formula
if (n==1),c=[1 1]/2; end
if (n==2),c=[1 2 1]/6; end
if (n==3),c=[1 3 3 1]/8; end
if (n==4),c=[7 32 12 32 7]/90; end
if (n==5),c=[19 75 50 50 75 19]/288; end
if (n==6),c=[41 216 27 272 27 216 41]/840; end
c=n*(t(2)-t(1))*c;

for i=1:n:N-n
   x(i,:)=c*x(i:i+n,:);        % store result in place
end

y=cumsum(x(1:n:N-n,:));
t=t(n+1:n:N);              % return the time scale

return
% -----------------------------
% End of 'integral.m'
% -----------------------------
```

5. SHAFT.M

In order to analyze the shaft encoder data that was recorded during the different motion programs.

```
function shaft(fname)

% ----------------------------------------------------------------
% function shaft(fname)
% ----------------------------------------------------------------
%
% M-File to analyze the shaft encoder readings recorded for SHEPHERD's
% motion according to the different motion profiles.
%
% Author:     Thorsten Leonardy
% Date:       11/11/97
% Compiler:   MATLAB V4.21c
%
% Input:      fname = name of data file (no extension '*.dat')
%             e.g. at the prompt >>shaft('linear4')
% ----------------------------------------------------------------

% load data
```

```
eval(['load -ascii ' fname '.dat, data=' fname ';' fname '=[];'])

% reshape the data
N=length(data)/8              % number of 10ms intervals contained in data
data=reshape(data,8,N);
t=0.01*(1:N);                 % the time base

driveDelta=data(1:2:8,:)';    % driving data [counts/10ms]
steerDelta=data(2:2:8,:)';    % steering data [counts/10ms]

% account for the fact that drive encoders for wheels 2 and 4 read negative
% differences if wheels are driving forward
driveDelta(:,2:2:4)=-driveDelta(:,2:2:4);

% accumulate the data to obtain true rotation of motors
drive=cumsum(driveDelta);     % the distance travelled
steer=cumsum(steerDelta);     % the angle steered

% scale to SI units
drive=drive/87914;   % drive distance in [m]
steer=steer/256;     % angle steered in degrees

% plot data
figure
for i=1:4
   if (mod(i,2))
      subplot(2,2,i+1)
   else
      subplot(2,2,i-1)
   end
   plot(t,drive(:,i)),grid
   title(['Wheel ' num2str(i)],'FontSize',8)
   xlabel('Time [sec]','FontSize',8)
   ylabel('Drive distance [m]','FontSize',8)
   set(gca,'FontSize',6,'Box','off')
   a=axis; a(3)=min(drive(:,i)); a(4)=max(drive(:,i)); axis(a)
end
eval(['print -dps2 shaft' num2str(gcf) '.ps'])

figure
plot(t,steer),grid
title('Steer values for Wheels 1..4 with steer value set to zero','FontSize',8)
xlabel('Time [sec]','FontSize',8)
ylabel('Steer angle [degrees]','FontSize',8)
set(gca,'FontSize',6,'Box','off')
ix=min(find(t>=65));
for i=1:4
   text(t(ix),steer(ix,i),['Wheel' num2str(i)],...
        'HorizontalAlign','left','VerticalAlign','top','FontSize',6)
end
eval(['print -dps2 shaft' num2str(gcf) '.ps'])


return
% ------------------------------------------------------------------
% end of 'shaft.m'
% ------------------------------------------------------------------
```

APPENDIX C: GCC COMPILER SOURCE-FILES

This appendix lists the C-source code that is being referred to throughout the text. Each individual source file was written in C and crosscompiled using the GCC Compiler Version 2.72 with the following command line:

```
gcc -c -m68040 -o filename.o filename.c
```

1. IMU.C

The file 'imu.c' provides all the routines required to implement the inertial measurement sensor as outlined in Chapter V. Moreover, they provide the interface for further development of the system.

```
/* ----------------------------------------------------------------------- *
 *                                                                         *
 * File:         I M U . C                                                 *
 *                                                                         *
 * Environment:  GCC Compiler v2.7.2                                       *
 * Last update:  10 September 1997                                         *
 * Name:         Thorsten Leonardy                                         *
 * Purpose:      Provides routines required for controlling the inertial   *
 *               measurement sensor.                                       *
 *                                                                         *
 * Compiled:     >gcc -c -m68040 -o navigat.o navigat.c                    *
 *                                                                         *
 * ----------------------------------------------------------------------- */

/* ----------------------------- R E A D M E ------------------------------

   Here is how the routines work:

   1. Make sure that initVME9325 is called inside main()
      this will setup the proper interrupt handling for reading data
      from the accelerometer.

   2. A/D-Block conversions as specified in initVME9325 will be initiated with every
      10ms timer interrupt. However, to make the data available, make sure that
      interrupt for conversion complete are being issued:

   3. Call startVME9325 to enable block conversion complete interrupts
      on IRQ-5 to 68040 processor and therefore copy data into main memory

   4. To seize copying data into main memory, call stopVME9325

   5. The A/D converter is setup such that after every 10ms timer interrupt
      a block conversion will be initiated. A total of AD_NUM_CONVERSIONS
      conversions will be performed on the four channels on the IMU
      in the sequence CH0, CH1, CH2, CH3, CH0, ...
      The sample time is set to be 25us (hence, one specific channel will
      be sampled every 100us)

   6. If interrupts are enabled, the most recent data obtained with every
      10ms timer interrupt will be stored in the structure imu as defined
      in SHEPHERD.H
```

```
    7. The boards status can be observed at the front panel:
       (a) green LED is on -> board performs A/D-Conversions, interrupts enabled
       (b) green LED is off -> board performs A/D-Conversions, interrupts disabled
       (c) red LED is toggling -> Interrupts are being handled by the handler,
                                  data is read from board into SHEPHERD main memory
       (d) red LED is on/off  -> interrupt handler is not being called

    ---------------------------------------------------------------------- */

#include "shepherd.h"
#include "imu.h"


int   adCounter;            /* counter for debugging purposes */
int   mainMemCounter;       /* to count the data stored in main memory */


/* the next is used as temporary storage for analyzing acceleration DATA */
unsigned short *mainMemData;


/* ---------------------------------------------------------------------- *
 * initVME9325(void)                                                      *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  24 July 1997                                             *
 * Name:         Thorsten Leonardy                                        *
 *                                                                        *
 * Purpose:      Initializes AD-Board VME9325. Board will convert         *
 *               analog data from channels specified and store the respec-  *
 *               tive digital data (2 Bytes per channel, 12 bit data, lowest *
 *               nibble is zero) sequentially in dual port ram.           *
 *                                                                        *
 *               Board will operate in Block mode with interrupts and timed  *
 *               periodic triggering (10ms cycle). E.g. perform 10 conver-  *
 *               sions on each of the four channels. Once 40 conversions are *
 *               made, initiate interrupt to read data into main memory and  *
 *               eventually smooth/filter data.                           *
 *                                                                        *
 * ---------------------------------------------------------------------- */
void initVME9325(void)
{

    unsigned char *ad = (unsigned char*) VME9325_BASE;        /* base address */
    unsigned char *vmeICR4 = (unsigned char*)VIC_IRQ4;        /* VME ICR IRQ-4*/
    long *vadr;                                       /* isr Vector base address */

    *(ad+0x81)=0x10;    /* software reset */
    *(ad+0x81)=0x02;    /* turn both  LEDs on to indicate board undergoes     */
                        /* initialization                                     */

    /* ---------------------------------------------------------------- *
     * Interrupt settings for VIC                                       *
     * ---------------------------------------------------------------- */
    vadr=(long*)0xffe40158;   /* VBA address for interrupt handler (4 * 0x56 = 0x158) */
    *vadr=(long)handlerVME9325;   /* write address of handler into Vector Table */

    /* set up VIC interface for VME-Bus interrupts to TUARUS. AD-Board asserts    */
    /* IRQ-4 upon interrupt to VME-Bus. Route as IRQ-2 to MC68040. CAUTION !!!    */
    /* make sure jumper J7 on AD-Board is set correctly !!!                       */
    *vmeICR4=0x82;  /* disable VME-Bus IRQ4 input, route as IRQ-2 to Processor */

    *(ad+0x83)=0x56;    /* interrupt vector number provided by board to VIC */

    /* program scan sequence (may wish to arrange channels to be scanned differently) */
    /* channels are scanned, converted and stored in memory in this order             */
    /*    *(ad+0x87)=0x00;    /* channel 0 (ax, +-7.5V input range, gain x1) */
    /*    *(ad+0x87)=0x01;    /* channel 1 (ay, +-7.5V input range, gain x1) */
    *(ad+0x87)=0x60;     /* channel 0 (ax, +-7.5V input range, gain x8) */
    *(ad+0x87)=0x61;     /* channel 1 (ay, +-7.5V input range, gain x8) */
    *(ad+0x87)=0x02;     /* channel 2 (az, +-7.5V input range, gain x1) */
    *(ad+0x87)=0xc3;     /* channel 3 (wy, +-2.5V input range, gain x4) */
                         /* gain x4 to cover max. input range +-10V,    */
                         /* set EOS bit to indicate end of scan sequence*/

```

```
    /* setup Board Control Register */
    *(ad+0x85)=0x08;    /* enable timer circuit, enable interrupts              */
                        /* block mode, software initiates very first trigger    */

    /* setup timed periodic triggering circuit for 50usec ( T = 10 * 10 / 2 MHz )*/
    *(ad+0x8f)=0x54;    /* setup counter to receive single byte prescaler count */
    *(ad+0x8b)=0x0A;    /* load prescaler value into Timer Prescaler Register   */
    *(ad+0x8f)=0x94;    /* setup counter to receive single byte timer count     */
    *(ad+0x8d)=0x0A;    /* load Conversion Timer Register                       */

    /* load conversion count register */
    *((unsigned short *)(ad+0x90))=200;

   /* initialization is complete */
    *(ad+0x81)=0x01;    /* turn off both LED, disable interrupts                */

    sioOut(0,"A/D-Board initialized\n\r");

    return;
}   /* end of AD_Init */


/* ------------------------------------------------------------------------ *
 * analyzeVME9325                                                           *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Last update:  24 July 1997                                               *
 * Name:         Thorsten Leonardy                                          *
 *                                                                          *
 * Purpose:      Saves the data for one complete block conversion cycle from *
 *               dual-port RAM of A/D-Board to Shepherd's main memory.      *
 *               In the future, this routine shall be utilized to analyze  *
 *               and filter the data and save only the filtered data.       *
 *               This is called from the interrupt handler routine          *
 *               AD_Handler.                                                *
 *                                                                          *
 * ------------------------------------------------------------------------ */
void analyzeVME9325(void)
{
   unsigned short *ad;  /* base address for data */
   int i;
   unsigned short adData[AD_NUM_CONVERSIONS];


   ad=(unsigned short*)VME9325_DATA;  /* load base address for dual port RAM */

   /* ------------------------------------------------------- *
    * here goes the filtering ...                             *
    * ------------------------------------------------------- */
    if ((adCounter%5)==0)
       toggleVME((unsigned char *)0xfd800000,0x01); /* toggle red LED every 50 msec*/

    adCounter++;

   /* ------------------------------------------------------- *
    * This is temporary backup                                *
    * ------------------------------------------------------- */

   for (i=0; i<AD_NUM_CONVERSIONS; i++) {
     adData[i]=*ad++;               /* neglect lower nibble */
    *mainMemData++=adData[i];     /* save data in main memory */
   }

#ifdef 0

   /* once data is filtered, store obtained values in imu */
   imu.ax=adData[0];
   imu.ay=adData[1];
   imu.az=adData[2];
   imu.omega_z=adData[3];

#endif

   /* reload start conversion register for next block conversion */
   ad=(unsigned short*)0xfd800090; /* address for SCR */
   *ad=AD_NUM_CONVERSIONS;         /* reload register */
```

```
   return;
}  /* end of analyzeVME9325 */


/* ---------------------------------------------------------------------------- *
 * startVME9325(void)                                                            *
 *                                                                               *
 * Environment:  GCC Compiler v2.7.2                                             *
 * Last update:  10 September 1997                                               *
 * Name:         Thorsten Leonardy                                               *
 *                                                                               *
 * Purpose:      enables interrupts issued by the VME9325 board.                 *
 *                                                                               *
 * Called from:  whatever function.                                              *
 *                                                                               *
 * ---------------------------------------------------------------------------- */
void startVME9325(void)
{
   unsigned char *statusRegister=(unsigned char *)VME9325_BASE+0x0081;
   unsigned char *vmeICR4 = (unsigned char*)VIC_IRQ4;        /* VME ICR IRQ-4*/

   /* initialize global variables ... */
    mainMemData=(unsigned short *)IMU_DATA_ADR;  /* start address for data storage */
    adCounter=0;                                 /* counter for debugging purposes */

   *vmeICR4=0x02;  /* enable VME-Bus IRQ4 input, route as IRQ-2 to Processor       */

   /* write status register to enable interrupt and turn off red LED               */
   *statusRegister=0x09;    /* turn off both LEDs, enable interrupts               */


   return;
}  /* end of startVME9325 */


/* ---------------------------------------------------------------------------- *
 * stopVME9325(void)                                                             *
 *                                                                               *
 * Environment:  GCC Compiler v2.7.2                                             *
 * Last update:  10 September 1997                                               *
 * Name:         Thorsten Leonardy                                               *
 *                                                                               *
 * Purpose:      disables interrupts off the VME9325 AD-Board. Yet, board        *
 *               will still perform A/D-Conversions but data will not be         *
 *               made available to the operating system.                         *
 * Called from:                                                                  *
 *                                                                               *
 * ---------------------------------------------------------------------------- */
void stopVME9325(void)
{
   unsigned char *statusRegister=(unsigned char *)VME9325_BASE+0x0081;
   unsigned char *vmeICR4 = (unsigned char*)VIC_IRQ4;   /* VME ICR IRQ-4*/

#ifdef 0
   /* initialize global variables ... */
    mainMemData=(unsigned short *)IMU_DATA_ADR;  /* start address for data storage */
    adCounter=0;                                 /* counter for debugging purposes */
#endif

   *vmeICR4=0x82;  /* disable VME-Bus IRQ4 input, route as IRQ-2 to Processor */

   /* write status register to disable interrupt and turn off red LED */
   *statusRegister=0x01;    /* turn off both LEDs, disable interrupts  */

   return;
}  /* end of stopVME9325 */



/******************************************************************************
  Assembler routines
 ******************************************************************************/
```

```
/* ---------------------------------------------------------------------- *
 * handlerVME9325                                                         *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  10 September 1997                                        *
 * Name:         Thorsten Leonardy                                        *
 *                                                                        *
 * Purpose:      Handles the VME-Bus interrupt request from the A/D-Board. *
 *                                                                        *
 * ---------------------------------------------------------------------- */


  asm("
      .even
      .text
      .globl _handlerVME9325

_handlerVME9325:


      link     a6,#-184          /* allocate 184 Bytes on stack to save registers    */
      fsave    a6@(-184)
#ifdef 0
      fmovemx  fp0-fp7,sp@-      /* move floating point registers 80 bit each        */
      fmovel   fpcr,sp@-         /* move floating point Control Regioster            */
      fmovel   fpsr,sp@-         /* move floating point status register              */
      fmovel   fpiar,sp@-        /* move floating point Instruction address register */
#endif
      moveml   d0-d7/a0-a5,sp@-  /* save data and address registers (14*4 Byte)      */

      addq.l   #1,_adCounter     /* increment counter (testing purpose only */

      move.l   #0xfd800081,a0    /* load address status register */
      and.b    #0xfd,(a0)        /* turn off green LED           */

      move.l   #0xfd800090,a0    /* reload start conversion register */
      move.w   #200,(a0)


#ifdef 1

      move.l   #0xfd820000,a0    /* load address for dual port RAM */
      lea      _mainMemData,a1
      move.l   (a1),a2

      clr.l    d0                /* loop counter */

_loop:
      cmp.l    #199,d0
      ble.b    _proceed
      nop
      bra.b    _done
      nop

_proceed:

      move.w   (a0),d1           /* read next two byte of dual port RAM
      nop                        /* caution: need this due to pipelining */
      move.w   d1,(a2)
      nop
      addq.l   #2,a0             /* increment pointer in dual port RAM */
      addq.l   #2,a2             /* increment pointer to next main memory location */
      addq.l   #1,d0             /* increment loop counter */
      bra.b    _loop

_done:

      move.l   a2,(a1)          /* write back the next main memory location */


 /*     jsr      _analyzeVME9325    /* copy data from A/D-Boards dual-port RAM to main  */
                                 /* memory and filter, analyze it                     */
#endif
```

```
        moveml   sp@+,d0-d7/a0-a5

#ifdef 0
        fmovel   sp@+,fpiar
        fmovel   sp@+,fpsr
        fmovel   sp@+,fpcr
        fmovemx  sp@+,fp0-fp7
#endif

        frestore a6@(-184)
        unlk     a6

        rte
  ");


/*****************************************************************************
  End of imu.c
 *****************************************************************************/

```

2. MOTOR.C

The file 'motor.c' provides the routines required to control the servo motors. Although the listing was already given by Mays/Reid [1], some changes had beend done to improve the overall execution time.

```
/* ==================================================================
// Edward Mays
// Shepherd project
// 20 February 1997
// update: 27 October 1997 Thorsten Leonardy
//          -> provide code to detect slip,
//          -> eliminate calls to readDriveEncoders, readSteerEncoders
//             by including code in readEncoders (improves execution speed)
//          -> compute speed and angular velocity immediately inside
//             readEncoders.
// MotionControl
// ================================================================*/


#include "shepherd.h"
#include "motor.h"
#include "movement.h"
#include "math.h"

double theta, omega, speed;
double a,                          /* acceleration in cm/sec^2 */
       dd[4];                      /* driveDelta required for velocity to steer */
int timeForTurn[8];                /* storage for time it took to rotate 360 degrees [10ms] */
short testSpeed=0x0b00;            /* temp variable for changing speed */
double radPerDigit[ARRAY_SIZE];
int ddc=10000,tc=2000;             /* desired vale for driveDelta */

int *leoData=(int *)0x00100000; /* start data storage */


void readEncoders() {
   readDriveEncoders(driveReadings);
   readSteerEncoders(steerReadings);
}


void readDriveEncoders(unsigned long int array[])
{
   unsigned char *p=(unsigned char*)VMECTR1, c1, c2, c3;
   int ix;
   long int temp;
```

```
   for (ix=0; ix<4; ix++) {   /* read all four motors subsequentially */

      *(p+3)=0x03;            /* load  output latch from counter */
      *(p+3)=0x01;            /* control register, initialize two-bit output latch */

      /* read three bytes for specific counter ix and save in status   */
      /* first access to Output Latch Register reads least significant */
      /* byte first                                                     */

      c1 = *(p+1) & 0x00ff;
      c2 = *(p+1) & 0x00ff;
      c3 = *(p+1) & 0x00ff;
      array[ix] = ((unsigned int)c1)| ((unsigned int)c2 << 8) |
              ((unsigned int)c3 << 16);

      p=p+4;                      /* increment pointer for next counter */


   }
   return;
} /* end of readDriveEncoders */


void readSteerEncoders(unsigned long int array[])
{
   unsigned char *p=(unsigned char*)(VMECTR1 + 0x0100), c1, c2, c3;
   int ix;


   for (ix=0; ix<4; ix++) {   /* read all four motors subsequentially */

      *(p+3)=0x03;            /* load  output latch from counter */
      *(p+3)=0x01;            /* control register, initialize two-bit output latch */


 /* read three bytes for specific counter ix and save in status */
 /* first access to Output Latch Register reads least significant byte first */

      c1 = *(p+1) & 0x00ff;
      c2 = *(p+1) & 0x00ff;
      c3 = *(p+1) & 0x00ff;
      array[ix] = ((unsigned int)c1)| ((unsigned int)c2 << 8) |
              ((unsigned int)c3 << 16);


      p=p+4;                                  /* increment pointer for next counter */

   }
   return;
} /* end of readSteerEncoders */



void computeActualRates()
{

int i;
double count,speed;

 for(i=0; i<=3; i++)
   {
  if(PreviousCountSpeed[i] == 99999999)  /* for derivative for speed  */
   actualSpeeds[i]= 0.0;
 else
   actualSpeeds[i]=
     (convertDifference((driveReadings[i] - PreviousCountSpeed[i]))
     *DigitToCmDrive[i])/DELTA_T;
 PreviousCountSpeed[i] = driveReadings[i];

 if(PreviousCountSteer[i] == 99999999)  /* for derivative for steering  */
   actualAngleRates[i]= 0.0;
 else
   actualAngleRates[i]=
       (convertDifference((steerReadings[i] - PreviousCountSteer[i]))
     *digitToRadSteer)/DELTA_T;
```

```
 PreviousCountSteer[i] =  steerReadings[i];
   }
}



void accumulateDriveSpeed()
{
 int i;

for(i=0;i<=3;i++){
   Display_Speeds[i] += actualSpeeds[i];
 }
 return;
}

void accumulateDriveSteer()
{
 int i;

for(i=0;i<=3;i++){
   Display_Steers[i] += 10*actualAngleRates[i];
   actualAngles[i]  += actualAngleRates[i]*DELTA_T;
}
 return;
}



/*********************************************************************************
   Function convertDifference() returns the difference between the new shaft
   encoder position and the old shaft encoder position. The shaft encoder values
   contain only 24 bits (0x000000-0xffffff). The routine adjusts for the trans-
   ition from 0xffffff to 0x000000 and vice versa.
*********************************************************************************/

int convertDifference(int value)
{
   if(value < -0x800000)
      value &= 0x00ffffff;
   else  if(value >= 0x800000)
      value |= 0xff000000;

   return value;
}


/* -------------------------------------------------------------------------- *
 * readNewEncoder()                                                           *
 *                                                                            *
 * Environment:  GCC Compiler v2.7.2                                          *
 * Name:         Thorsten Leonardy                                            *
 * Last update:  10/27/97                                                     *
 * Purpose:      This function reads the counter status for drive and steer   *
 *               motors every 10ms and stores the current values in the       *
 *               variables 'driveReadings' and 'steerReadings'. In addition,  *
 *               the incremental change to the last update is stored in the   *
 *               variables 'driveDelta' and 'steerDelta' to allow for compu-  *
 *               ting the most current speeds and angular velocities.        *
 *                                                                            *
 * Called from:  driver() in movement.c                                       *
 * -------------------------------------------------------------------------- */
void readNewEncoder()
{

   unsigned char *p,*d;
   int ix;

   p=(unsigned char*)VMECTR1;  /* access steering counter registers          */

   for (ix=0; ix<4; ix++) {     /* read all four driving motors sequentially  */

      driveCountPrevious[ix]=driveCount[ix]; /* save previous value           */
      steerCountPrevious[ix]=steerCount[ix]; /* save previous value           */

      /* -------------------------------- */
```

```
      /* read drive encoders for wheel ix */
      /* -------------------------------- */
      *(p+3)=0x03;                          /* load  output latch from counter */
      *(p+3)=0x01;                          /* initialize two-bit output latch */

      d=((unsigned char*)&driveCount[ix])+2; /* start with LSB, need offset    */
      *d-- = *(p+1) & 0x00ff;               /* read LSB first                  */
      *d-- = *(p+1) & 0x00ff;               /* read next byte                  */
      *d   = *(p+1) & 0x00ff;               /* read most significant byte      */

      /* -------------------------------- */
      /* read steer encoders for wheel ix */
      /* -------------------------------- */
      *(p+0x103)=0x03;                      /* load  output latch from counter */
      *(p+0x103)=0x01;                      /* initialize two-bit output latch */

      d=((unsigned char*)&steerCount[ix])+2; /* load LSB first                 */
      *d-- = *(p+0x101) & 0x00ff;           /* read LSB first                  */
      *d-- = *(p+0x101) & 0x00ff;           /* read next byte                  */
      *d   = *(p+0x101) & 0x00ff;           /* read most significant byte      */

      p=p+4;                                /* increment pointer for next motor*/

      /* determine difference between previous and current encoder reading */
      steerDelta[ix]=(steerCount[ix]-steerCountPrevious[ix])/256;
      driveDelta[ix]=(driveCount[ix]-driveCountPrevious[ix])/256;

      /* consider the fact that a positive driveDelta for wheels 2 and 4 */
      /* indicate that wheel is driving backwards !!! Thgus, change sign */
      driveDelta[ix]=(driveCount[ix]-driveCountPrevious[ix])/256;

      /* the following is just for testing purposes [leo, 11/17/97] */
      *encoderData++=driveDelta[ix];    /* store in main memory */
      *encoderData++=steerDelta[ix];    /* store in main memory */

   } /* end of for */

   /* account for the fact that a positive driveDelta for wheels 2 and 4 */
   /* indicate that wheel is driving backwards !!! Thus, change sign to  */
   /* obtain a positive driveDelta for wheel driving forward !!!         */
   driveDelta[1]=-driveDelta[1];
   driveDelta[3]=-driveDelta[3];

   return;

} /* end of readNewEncoder */



/* ------------------------------------------------------------------------ *
 * readNewEncoder()                                                         *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Name:         Thorsten Leonardy                                          *
 * Last update:  10/27/97                                                   *
 * Purpose:      This function reads the counter status for drive and steer *
 *               motors every 10ms and stores the current values in the     *
 *               variables 'driveReadings' and 'steerReadings'. In addition, *
 *               the incremental change to the last update is stored in the *
 *               variables 'driveDelta' and 'steerDelta' to allow for compu- *
 *               ting the most current speeds and angular velocities.       *
 *                                                                          *
 * Called from:  driver() in movement.c                                     *
 * ------------------------------------------------------------------------ */
void readEncoder()
{

   unsigned char *p,*d;
   int ix;

   p=(unsigned char*)VMECTR1;  /* access steering counter registers          */

   for (ix=0; ix<4; ix++) {     /* read all four driving motors sequentially   */

      driveCountPrevious[ix]=driveCount[ix]; /* save previous value           */
      steerCountPrevious[ix]=steerCount[ix]; /* save previous value           */
```

```

        /* -------------------------------- */
        /* read drive encoders for wheel ix */
        /* -------------------------------- */
        *(p+3)=0x03;                          /* load  output latch from counter */
        *(p+3)=0x01;                          /* initialize two-bit output latch */

        d=((unsigned char*)&driveCount[ix])+2; /* start with LSB, need offset    */
        *d-- = *(p+1) & 0x00ff;               /* read LSB first                  */
        *d-- = *(p+1) & 0x00ff;               /* read next byte                  */
        *d   = *(p+1) & 0x00ff;               /* read most significant byte      */

        /* -------------------------------- */
        /* read steer encoders for wheel ix */
        /* -------------------------------- */
        *(p+0x103)=0x03;                      /* load  output latch from counter */
        *(p+0x103)=0x01;                      /* initialize two-bit output latch */

        d=((unsigned char*)&steerCount[ix])+2; /* load LSB first                 */
        *d-- = *(p+0x101) & 0x00ff;           /* read LSB first                  */
        *d-- = *(p+0x101) & 0x00ff;           /* read next byte                  */
        *d   = *(p+0x101) & 0x00ff;           /* read most significant byte      */

        p=p+4;                                /* increment pointer for next motor*/

        /* determine difference between previous and current encoder reading */
        steerDelta[ix]=(steerCount[ix]-steerCountPrevious[ix])/256;
        driveDelta[ix]=(driveCount[ix]-driveCountPrevious[ix])/256;

    } /* end of for */

    /* account for the fact that a positive driveDelta for wheels 2 and 4 */
    /* indicate that wheel is driving backwards !!! Thus, change sign to  */
    /* obtain a positive driveDelta for wheel driving forward !!!         */
    driveDelta[1]=-driveDelta[1];
    driveDelta[3]=-driveDelta[3];

    return;

} /* end of readEncoder */



/* ---------------------------------------------------------------------- *
 * computeSpeedAndAngle()                                                 *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Name:         Thorsten Leonardy                                        *
 * Last update:  11/21/97                                                 *
 * Purpose:      This function computes the speeds, angles and angular velo- *
 *               city for all four wheels based on the most recent shaft  *
 *               encoder readings from readNewEncoder().                  *
 *                                                                        *
 * Called from:  driver() in movement.c                                   *
 * ---------------------------------------------------------------------- */
void computeSpeedAndAngle(void)
{

  int i;

  /* compute measured driving speed [cm/sec] and steering angle [rad] and */
  /* steering rate [rad/sec].                                             */
  for(i=0; i<=3; i++) {
     actualSpeeds[i]     = ((double)driveDelta[i])*CM_PER_DIGIT/0.01;
     actualAngles[i]    += ((double)steerDelta[i])*RAD_PER_DIGIT;
     actualAngleRates[i] = ((double)steerDelta[i])*RAD_PER_DIGIT/0.01;
  }
  return;
}



/*                                                       */
/* Verifies validity of incoming speeds/angles and converts */
/* digitial input for the DA board                       */
/*                                                       */
```

```
void driveMotors(){

    int ix,Speed_Digit,Steer_Digit, counter;
    double speed1, steer1, temp;

    unsigned short bitMask=0x8000;       /* access bit 15 for align  wheel 1 */
    unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */

    bitMask = bitMask >> 3;

    /*  updateWheelDrive();  wheel values for driving              */
    /*  updateWheelSteer();                                        */
    /*  comupte the current actual wheel direction in WheelDirAct[] */

    if (mode != 100){
      for(ix =0; ix <ARRAY_SIZE; ix++){
        /* ********************steering/driving interaction*************     */
        /* here +/- 1/50 of the steering value is added to the driving       */
        /* for each specified wheel. Note the negative sign on elements [1]  */
        /* and [3]provide the same direction driving as elements [0] and [2] */

        Omega_Speed = desiredSpeeds[ix] +
         SteerDriveInteract*desiredAngleRates[ix]*18.9; /* cm/sec */

        /* conversion to digits   */
        Speed_Digit = velocityReferenceTable(Omega_Speed,ix) +
   DriveFeedBackGain*(Omega_Speed - actualSpeeds[ix]);
        Steer_Digit = rateReferenceTable(desiredAngleRates[ix])
 + steerFeedbackGain*(desiredAngleRates[ix]-actualAngleRates[ix])
 + angleFeedbackGain*norm(desiredAngles[ix]-actualAngles[ix]);

        if (Speed_Digit>DigitsHigh)         /* Limitation  */
          Speed_Digit= DigitsHigh;
        if (Steer_Digit>DigitsHigh)
          Steer_Digit= DigitsHigh;
        if (Speed_Digit<DigitsLow)
          Speed_Digit= DigitsLow;
        if (Steer_Digit<DigitsLow)
          Steer_Digit= DigitsLow;

       switch(mode){
 case 2:
 case 3:
 case 4:
 case 5:
 case 6:
 case 7:
 case 8:
 case 9:
 case 10:
         case 11:   /* case 11: linear test drive, added 11/03/97 Leo */
   speedDigits[ix]= (short)Speed_Digit;  /* casting to short */
   steerDigits[ix]= (short)Steer_Digit;
   break;

 case 1:
   speed1 = speedDigits[ix];
   steer1 = steerDigits[ix];
   if ( speed1 > 0) speed1--;
   if ( speed1 < 0) speed1++;
   if ( steer1 > 0) steer1--;
   if ( steer1 < 0) steer1++;
   speedDigits[ix] = speed1;
   steerDigits[ix] = steer1;
     break;
       } /* end switch */
      } /* end for */
     } /* end if */
     else {
        for (ix=0; ix<3; ix++){
   steerDigits[ix] = 0;
}
for (ix=0; ix<4; ix++){
   speedDigits[ix] = 0;
}
```

```
switch(modeTstate){
 case 0:
    steerDigits[3] = 50*Flag;
    modeTstate = 1;
    break;

 case 1:
    modeTstate = 2;
    break;

 case 2:
    modeTstate = 3;
    break;

 case 3:
    modeTstate = 4;
    break;

 case 4:
    modeTstate = 5;
    break;

 case 5:
    modeTstate = 6;
    break;

 case 6:
    modeTstate = 7;
    break;


 case 7:
    modeTstate = 8;
    break;

 case 8:
    modeTstate = 9;
    break;

 case 9:
    modeTstate = 10;
    break;

 case 10:
    modeTstate = 11;
    break;

 case 11:
    modeTstate = 12;
    break;

 case 12:
    modeTstate = 13;
    break;

 case 13:
    modeTstate = 14;
    break;

 case 14:
    modeTstate = 15;
    break;

 case 15:
    modeTstate = 16;
    break;

 case 16:
    modeTstate = 17;
    break;

 case 17:
    modeTstate = 18;
    break;

 case 18:
```

```
        modeTstate = 19;
        break;

    case 19:
        if (bitMask&*servoStatus)/* read servo status, */
{                           /*wait until wheel aligned */
            Flag = -Flag;
            modeTstate = 20;
}
        break;

    case 20:
        steerDigits[3] = 0;
        modeTstate = 21;
        break;

    case 21:
        modeTstate = 22;
        break;

    case 22:
        modeTstate = 23;
        break;

    case 23:
        modeTstate = 24;
        break;

    case 24:
        modeTstate = 25;
        break;

    case 25:
        modeTstate = 26;
        break;

    case 26:
        modeTstate = 27;
        break;

    case 27:
        modeTstate = 0;
        break;

    default :  break;
 } /* end switch */
        } /* end else */

#ifdef 0
      driveSteer(steerDigits);
      driveSpeeds(speedDigits);
#endif

      /* here is a more efficient way of setteing the speeds [Leo, 11/18/97] */
      /* instead of using the functions driveSteer and driveSpeeds ...       */
      setServoSpeed();


}/* end driveMotors */



double velocityReferenceTable(double desiredVelocity,int i)
{
    double inVelocity,
           outVelocity;

    inVelocity=new_abs(desiredVelocity);

     if (inVelocity>=0.0 && inVelocity<=5.0)
       outVelocity = inVelocity*K1[i];

     if (inVelocity>5.0 && inVelocity< 8.0)
       outVelocity = inVelocity*K2[i];

     if (inVelocity>=8.0 && inVelocity<20.0)
```

```
      outVelocity = inVelocity*K3[i];

    if (inVelocity>=20.0 && inVelocity<= 70.0)
      outVelocity = inVelocity*K4[i];

   if (inVelocity>70.0 && inVelocity<K5)
      outVelocity = inVelocity*K6[i];

    if (inVelocity> K5)
     outVelocity=1023;

    if (desiredVelocity< 0.0)
      outVelocity = - outVelocity;

   return outVelocity;
} /* end velocityLookupTable */


double rateReferenceTable(double desiredRate)
{
   double inRate,
          outDigit;

   /*outDigit = new_abs(desiredRate); *//* test only */

    inRate=new_abs(desiredRate);

    if (inRate<= 5.234)
      outDigit = inRate*195.4155 ;
    else
       outDigit=1023;


    if (desiredRate< 0.0)
      outDigit = - outDigit;

   return outDigit;
}



/* ---------------------------------------------------------------------- *
 * readOneEncoder()                                                       *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Name:         Thorsten Leonardy                                        *
 * Last update:  10/27/97                                                 *
 * Purpose:      Reads only the encoder specified by 'wheel':             *
 *               wheel = 0 ... 3 reads drive encoder for wheel 1..4       *
 *               wheel = 4 ... 7 reads steer encoder for wheel 1..4       *
 * Note:         !!! The data (24 bit) is still left adjusted !!!         *
 * ---------------------------------------------------------------------- */
void readOneEncoder(int ix, int *data)
{

   unsigned char *p,*d;

   p=(unsigned char*)VMECTR1;           /* access steering register       */
   p=p+4*ix;
   if (ix>3) p=p+0x0090;     /* account for the fact VMECTR2=VMCTR1+0x100 */

   *(p+3)=0x03;                         /* load  output latch from counter */
   *(p+3)=0x01;                         /* initialize two-bit output latch */

   d=(unsigned char *)data;             /* start with LSB, need offset     */
   d=d+2;
   *d-- = *(p+1) & 0x00ff;            /* read LSB first                 */
   *d-- = *(p+1) & 0x00ff;            /* read next byte                 */
   *d   = *(p+1) & 0x00ff;            /* read most significant byte     */

   return;

} /* end of readOneEncoder */


/* ---------------------------------------------------------------------- *
```

```
 * linearMotion()                                                        *
 *                                                                       *
 * Environment:  GCC Compiler v2.7.2                                     *
 * Name:         Thorsten Leonardy                                       *
 * Last update:  10/27/97                                                *
 * Purpose:      IMplements a linear motion test profile such that the   *
 *               vehicle is following steps in successive 10sec time     *
 *               intervals.                                              *
 * Call:         User presses 'l' on the keyboard (see user() in file user.c)*
 * ---------------------------------------------------------------------- */
void linearMotion1(void)
{
   double v1x, v1y, v2, v1yv1xRatio,omega2,omega3, beta,ro,ro2,wheelAngleV;
   int ix,Speed_Digit,Steer_Digit;
   short *servoOut;

   /* read all shaft encoders */
   readNewEncoder();

   /* compute the actual rates, velocities and angles */
   for (ix=0; ix<4; ix++){
      driveSpeed[ix]=driveDelta[ix]*CM_PER_DIGIT/DELTA_T;     /* [cm/s] */
      steerRate[ix]=steerDelta[ix]/DELTA_T;
      steerAngle[ix]=steerAngle[ix]+steerDelta[ix]*RAD_PER_DIGIT;
   } /* end of for ... */


   /* initialize temporary variables */
   speed=motion.Speed;
   theta=motion.Theta;
   omega=motion.Omega;

   /* ------------------------------------------------------------------- *
    * body motion (former in movement.c)                                  *
    * ------------------------------------------------------------------- */

   a=2.0;   /* acceleration is 2cm/sec^2 */

   if (time<1000) {
      speed=a*time/100.0;  /* rise linearly from 0 ..20 cm/sec in 10 secs */
   }

   if (time==1000){
      speed=a*10.0;     /* vehicle speed constant for next 10 sec */
   }

   if (time>=2000)
      if (time<3000)
          speed=a*(3000-time)/100.0;    /* decelerate to zero speed for 20sec..30sec */

   if (time==3000){
      speed=0.0;     /* stop vehicle for 30sec..40sec */
   }

   if (time>=4000)
      if (time<5000)
         speed=a*(4000.0-time)/100.0;   /* reverse motion, move back for 40sec .. 50sec */

   if (time==5000){
      speed=-a*10.0;     /* move back with constant velocity */
   }

   if (time>=6000)
      if (time<7000)
         speed=a*(time-7000.0)/100.0;

   if (time==7000){
      mode=0;
      stopVME9325();     /* stop A/D-Board */
      allOffAndZero();
   }

   /* compute required derivatives */
   speedDot=(speed-motion.Speed)/DELTA_T;
   thetaDot=(theta-motion.Theta)/DELTA_T;
   omegaDot=(omega-motion.Omega)/DELTA_T;
```

```
   /* update the motion */
   motion.Speed = speed;
   motion.Theta = theta;
   motion.Omega = omega;

   /* update the vehicle configuration */
   vehicle.heading = vehicle.heading + motion.Omega*DELTA_T;
   vehicle.coord.x = vehicle.coord.x + motion.Speed*DELTA_T * cos(motion.Theta);
   vehicle.coord.y = vehicle.coord.y + motion.Speed*DELTA_T * sin(motion.Theta);

   /* ---------------------------------------------------------------- *
    * drive motors (former in motor.c)                                  *
    * ---------------------------------------------------------------- */

    dd[0]=speed/wheelRadius[0]*16615.776;
    dd[1]=speed/wheelRadius[1]*16615.776;
    dd[2]=speed/wheelRadius[2]*16615.776;
    dd[3]=speed/wheelRadius[3]*16615.776;


    speedDigits[0]=(short)(0.0132421*dd[0]-1.15119);   /* set speed for wheel 1 */
    speedDigits[1]=(short)(0.0132276*dd[1]-1.17617);   /* set speed for wheel 2 */
    speedDigits[2]=(short)(0.0132283*dd[2]+0.17110);   /* set speed for wheel 3 */
    speedDigits[3]=(short)(0.0132680*dd[2]+1.21652);   /* set speed for wheel 4 */

    /* set the speeds */
    setServoSpeed();

  return;
}  /* end of leoMotion() */

void linearMotion2(void)
{
   double v1x, v1y, v2, v1yv1xRatio,omega2,omega3, beta,ro,ro2,wheelAngleV;
   int ix,Speed_Digit,Steer_Digit;
   short *servoOut;

   /* read all shaft encoders */
   readNewEncoder();

   /* compute the actual rates, velocities and angles */
   for (ix=0; ix<4; ix++){
      driveSpeed[ix]=driveDelta[ix]*CM_PER_DIGIT/DELTA_T;      /* [cm/s] */
      steerRate[ix]=steerDelta[ix]/DELTA_T;
      steerAngle[ix]=steerAngle[ix]+steerDelta[ix]*RAD_PER_DIGIT;
   } /* end of for ... */


   /* initialize temporary variables */
   speed=motion.Speed;
   theta=motion.Theta;
   omega=motion.Omega;

   /* ---------------------------------------------------------------- *
    * body motion (former in movement.c)                                *
    * ---------------------------------------------------------------- */

   a=100.0;   /* max acceleration [cm/sec^2] */

   /* no acceleration for t<1sec */

   if ((time>=100)&&(time<200))
      speed=0.005*(time-100)*(time-100); /* vehicle speed [cm/sec] (max is 50cm/sec */

   if ((time>=300)&&(time<400))
         speed=800.0+0.005*time*(time-800.0);

   if (time==400){
      mode=0;
      stopVME9325();    /* stop A/D-Board */
      allOffAndZero();
   }

   /* compute required derivatives */
   speedDot=(speed-motion.Speed)/DELTA_T;
```

```
    thetaDot=(theta-motion.Theta)/DELTA_T;
    omegaDot=(omega-motion.Omega)/DELTA_T;

    /* update the motion */
    motion.Speed = speed;
    motion.Theta = theta;
    motion.Omega = omega;

    /* update the vehicle configuration */
    vehicle.heading = vehicle.heading + motion.Omega*DELTA_T;
    vehicle.coord.x = vehicle.coord.x + motion.Speed*DELTA_T * cos(motion.Theta);
    vehicle.coord.y = vehicle.coord.y + motion.Speed*DELTA_T * sin(motion.Theta);

    /* ------------------------------------------------------------------ *
     * drive motors (former in motor.c)                                   *
     * ------------------------------------------------------------------ */

     dd[0]=speed/wheelRadius[0]*16615.776;
     dd[1]=speed/wheelRadius[1]*16615.776;
     dd[2]=speed/wheelRadius[2]*16615.776;
     dd[3]=speed/wheelRadius[3]*16615.776;


     speedDigits[0]=(short)(0.0132421*dd[0]-1.15119);   /* set speed for wheel 1 */
     speedDigits[1]=(short)(0.0132276*dd[1]-1.17617);   /* set speed for wheel 2 */
     speedDigits[2]=(short)(0.0132283*dd[2]+0.17110);   /* set speed for wheel 3 */
     speedDigits[3]=(short)(0.0132680*dd[2]+1.21652);   /* set speed for wheel 4 */

     /* set the speeds */
     setServoSpeed();

    return;
}  /* end of leoMotion2() */


/* ------------------------------------------------------------------------ *
 * setServoSpeed()                                                          *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Name:         Thorsten Leonardy                                          *
 * Last update:  10/27/97                                                   *
 * Purpose:      This function sets the speed as specified in global vars   *
 *               speedDigits and steerDigits to all servo motors.           *
 * Called from:  driver() in movement.c                                     *
 * ------------------------------------------------------------------------ */
void setServoSpeed(void)
{

 short *servoOut=(unsigned short*)(VME9210+0x0082);    /* Analog out     */

 *servoOut++= -speedDigits[0]*16;     /* set speed for driving wheel 1 */
 *servoOut++=  speedDigits[1]*16;     /* set speed for driving wheel 2 */
 *servoOut++= -speedDigits[2]*16;     /* set speed for driving wheel 3 */
 *servoOut++=  speedDigits[3]*16;     /* set speed for driving wheel 4 */

 *servoOut++=  steerDigits[0]*16;     /* set speed for driving wheel 1 */
 *servoOut++=  steerDigits[1]*16;     /* set speed for driving wheel 2 */
 *servoOut++=  steerDigits[2]*16;     /* set speed for driving wheel 3 */
 *servoOut++=  steerDigits[3]*16;     /* set speed for driving wheel 4 */

  return;
} /* End of setServoSpeed */



/* ------------------------------------------------------------------------ *
 * clearEncoder(motors)                                                     *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Last update:  03 November 1997                                           *
 * Name:         Thorsten Leonardy                                          *
 * Purpose:      This function clears all shaft encoders.                   *
 *                                                                          *
 * motors        bit mask to select motors, eg. 0x042 selects motor 2 and 7 *
 *               to be cleared.                                             *
 * ------------------------------------------------------------------------ */
```

```
void clearEncoder(unsigned char motors)
{
   unsigned char *p=(unsigned char*)VMECTR1;
   int ix;

   for (ix=0; ix<4; ix++,motors/=2) {
      if (motors & 0x01) *(p+3)=0x04;        /* clear respective counter */
      if (motors & 0x10) *(p+0x0103)=0x04;   /* clear steering counter   */
      p=p+4;                                 /* access next pointer      */
   }
   return;
} /* end of clearEncoder */


/* ---------------------------------------------------------------------- *
 * align()                                                                *
 * Environment:  GCC Compiler                                             *
 * Last update:  07 August 1997                                           *
 * Name:         Thorsten Leonardy and Yutaka Kanayama                    *
 * Purpose:      This function will align SHEPHERD's wheels such that all *
 *               will point in the forward direction. It utilizes the hall *
 *               sensors for each of the four wheels. All wheels are being *
 *               aligned simultaneously rather than successively.         *
 *                                                                        *
 * ---------------------------------------------------------------------- */
void align(void)
{
   unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);      /* Analog out */
   unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca);  /* digital input */
   unsigned int   *servoControl=(unsigned int *)VME2170;            /* Data Out */
   int ix;
   unsigned short bitMask,speed=0x0200;

   /* set steering speeds */
   *servoOut=-speed;        /* wheel1 -> rotate CW */
   *(servoOut+1)= speed;    /* wheel2 -> rotate CCW */
   *(servoOut+2)= speed;    /* wheel3 -> rotate CCW */
   *(servoOut+3)=-speed;    /* wheel4 -> rotate CW  */

   bitMask=0xf000;

   while(bitMask){
     if ( 0x8000 & *servoStatus ){
        *servoOut=0x0000;         /* set speed=0 for wheel 1 */
        bitMask=bitMask & 0x7000;
     }
     if ( 0x4000 & *servoStatus ){
        *(servoOut+1)=0x0000;   /* set speed=0 for wheel 2 */
        bitMask=bitMask & 0xb000;
     }
     if ( 0x2000 & *servoStatus ){
        *(servoOut+2)=0x0000;   /* set speed=0 for wheel 3 */
        bitMask=bitMask & 0xd000;
     }
     if ( 0x1000 & *servoStatus ){
        *(servoOut+3)=0x0000;   /* set speed=0 for wheel 4 */
        bitMask=bitMask & 0xe000;
     }
   }

   sioOut(0,"Aligned ...\n\r");

   return;
}  /* end of align */


/* ----------------------------------------------------- *
 * all servos on and set zero speed, [added 11/06/97, Leo] *
 * ----------------------------------------------------- */
void allOnAndZero(void){
   unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
   short *servoOut=(unsigned short*)(VME9210+0x0082);   /* Analog out driving wheel1 */
   int ix;

   for (ix=0; ix<8; ix++) *servoOut++=0x0000;   /* set zero speed */

```

```
     *servoControl=0x00924924;        /* turn on all motors */

    return;
} /* end of allOnAndZero */


/* ------------------------------------------------------- *
 * all servos off and set zero speed, [added 11/05/97, Leo] *
 * ------------------------------------------------------- */
void allOffAndZero(void){
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
    short *servoOut=(unsigned short*)(VME9210+0x0082);   /* Analog out driving wheel1 */
    int ix;

    for (ix=0; ix<8; ix++) *servoOut++=0x0000;   /* set zero speed */

     *servoControl=0x00000000;        /* turn on all motors */

    return;
} /* end of allOffAndZero */


/* ---------------------------------------- *
 * Set all driving motors to specific speed *
 * ---------------------------------------- */
void allDrive(short digit){
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
    short *servoOut=(unsigned short*)(VME9210+0x0082);   /* Analog out driving wheel1 */
    int ix;

    for (ix=0; ix<4; ix++) *servoOut++=digit;   /* set zero speed */

    *servoControl=0x00000924;        /* turn on driving motors */

    return;
} /* end of allDrive */


/* ----------------------------------------- *
 * Set all steering motors to specific speed *
 * ----------------------------------------- */
void allSteer(short digit)
{
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
    short *servoOut=(unsigned short*)(VME9210+0x008A);   /* Analog out steering wheel1 */
    int ix;

    for (ix=0; ix<4; ix++) *servoOut++=digit;   /* set zero speed */

    *servoControl=0x00924000;        /* turn on steering motors */

    return;
} /* end of allSteer */


/* --------------------------------------------- *
 * switches all motors off [added 11/05/97, Leo] *
 * --------------------------------------------- */
void allMotorsOff(void){
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */

     *servoControl=0x00000000;        /* turn off all motors */

    return;
} /* end of allMotorsOff */


/* -------------------------------------------- *
 * switches all motors on [added 11/05/97, Leo] *
 * -------------------------------------------- */
void allMotorsOn(void){
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */

     *servoControl=0x00924924;        /* turn on all motors */

    return;
```

```
} /* end of allMotorsOn */


/* ---------------------------------------------------------------------- *
 * driveTest()                                                            *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  29 October 1997                                          *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      This function computes the actual servo data for all     *
 *               driving motors.                                          *
 * Called from:  user() upon keyboard interaction (type 'd')              *
 * ---------------------------------------------------------------------- */
void driveTest()
{
   unsigned int   *servoControl=(unsigned int *)VME2170;          /* Data Out */
   unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);    /* Analog out */
   unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */
   unsigned short bitMask=0x8000;       /* access bit 15 for align  wheel 1 */
   unsigned char *p;
   unsigned int wheelSelect;
   int ix;

   *servoControl=0x00000000;            /* disable (turn off) all wheels    */

   servoOut=(unsigned short*)(VME9210+0x0082);    /* Analog out for drive wheel 1*/
   wheelSelect=0x00000004;                      /* select servo for driving wheel 1 */

   p=(unsigned char*)VMECTR1;

   for (ix=0; ix<4; ix++) {

      *servoOut=testSpeed;           /* set output value for servo first        */
      *servoControl=wheelSelect;     /* turn on selected servo motor            */

      sioOut(0,"Press '.' to start recording time\n\r");

      while (key!='.') ;             /* wait until user starts */

      *(p+3)=0x04;                   /* clear counter for driving wheel ix */

      readOneEncoder(ix,(int *)&driveCountPrevious[ix]); /* update encoder */
      readOneEncoder(ix,(int *)&steerCountPrevious[ix]); /* update encoder */

      timeForTurn[ix]=intCounter;    /* store time (start observing) */

      sioOut(0,"Press ',' to stop recording time\n\r");

      while (key!=',') ;              /* wait until user stops the process */

      timeForTurn[ix]=intCounter-timeForTurn[ix];

      *servoOut++=0x0000;            /* stop wheel */

      readOneEncoder(ix,(int *)&driveCount[ix]); /* update encoder */
      readOneEncoder(ix,(int *)&steerCount[ix]); /* update encoder */

      driveDelta[ix]=(driveCount[ix]-driveCountPrevious[ix])/256;
      steerDelta[ix]=(steerCount[ix]-steerCountPrevious[ix])/256;

      wheelSelect= wheelSelect<<3;     /* select next servo (motor)        */

   }

   *servoControl=0x00000000;            /* disable (turn off) all wheels    */

   return;
} /* end of driveTest */


/* ---------------------------------------------------------------------- *
 * velocityTest()                                                         *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  07 November 1997                                         *
 * Name:         Thorsten Leonardy                                        *
```

```
 * Purpose:     This function obtaines the velocity versus digit curve.    *
 *              Drive servos are given different velociies (digit) every   *
 *              two seconds. The first second is to obtain steady state, the*
 *              second second will record the shaft encoder difference, thus*
 *              giving rise to a encoder reading versus velocity curve.    *
 *              The commanded velocity goes from 500 .. -510 at present.   *
 *                                                                          *
 * Called from:  user() upon keyboard interaction (type 'v')               *
 * ------------------------------------------------------------------------ */
void velocityTest(void)
{
   unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
   short *servoOut=(unsigned short*)(VME9210+0x0082);   /* Analog out driving wheel1 */

   short speed,digit;

   speed=500;
   digit=speed*16;

   leoData=(int *)0x00100000;    /* start data storage */

   sioOut(0,"velocityTest\n\r");
   align();
   allOffAndZero();

   *servoControl=0x00000924;         /* turn on driving motors */

   readNewEncoder();
   time=0;                /* this will be altered by timer interrupt */

   /* set new driving values */
   *servoOut++=-digit;   /* set speed for wheel 1 */
   *servoOut++= digit;   /* set speed for wheel 2 */
   *servoOut++=-digit;   /* set speed for wheel 3 */
   *servoOut++= digit;   /* set speed for wheel 4 */

   while (speed>-510) {

      servoOut=(short *)(VME9210+0x0082);

      /* set new driving values */
      *servoOut++=-digit;   /* set speed for wheel 1 */
      *servoOut++= digit;   /* set speed for wheel 2 */
      *servoOut++=-digit;   /* set speed for wheel 3 */
      *servoOut++= digit;   /* set speed for wheel 4 */

      speed=speed-10;
      digit=speed*16;   /* shift nibble left */
      time=0;

      /* wait a second for motors to settle */
      while(time<100) ;

      readNewEncoder();

      /* record for a second */
      while(time<200) ;

      readNewEncoder();


     /* store the counter data for previous speed */
      *leoData++=steerDelta[0];
      *leoData++=steerDelta[1];
      *leoData++=steerDelta[2];
      *leoData++=steerDelta[3];
      *leoData++=driveDelta[0];
      *leoData++=driveDelta[1];
      *leoData++=driveDelta[2];
      *leoData++=driveDelta[3];
   }

   allOffAndZero();

   return;
} /* end of velocityTest */
```

```
/* ---------------------------------------------------------------------- *
 * circumferenceTest()                                                    *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  07 November 1997                                         *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      This function drives the vehicle in a straight line and  *
 *               stores the difference for all shaft encoders for a given *
 *               observation time. If the distance travelled is being     *
 *               measured, one can obtain the relation between shaft encoder *
 *               readings and wheel diameter.                             *
 * Called from:  user() upon keyboard interaction (type 'c')              *
 * ---------------------------------------------------------------------- */
void circumferenceTest(void)
{
    unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out */
    short *servoOut=(unsigned short*)(VME9210+0x0082);   /* Analog out driving wheel1 */

    short speed,digit;

    speed=300;
    digit=speed*16;

    leoData=(int *)0x00100000;    /* start data storage */

    sioOut(0,"circumferenceTest()\n\r");

    align();
    allOffAndZero();

    *servoControl=0x00000924;          /* turn on driving motors */

    /* determine the digits to command based on linea4r relationship obtained *
     * in velocityTest for each wheel individually.                          */

    /* assume for one second, that driveDelta=10000 */


    /* set new driving values for driveDelta approx 10000 over 1 sec */
    *servoOut++=(short)(-16*(0.0132421*ddc-1.15119));   /* set speed for wheel 1 */
    *servoOut++=(short)( 16*(0.0132276*ddc-1.17617));   /* set speed for wheel 2 */
    *servoOut++=(short)(-16*(0.0132283*ddc+0.17110));   /* set speed for wheel 3 */
    *servoOut++=(short)( 16*(0.0132680*ddc+1.21652));   /* set speed for wheel 4 */

    time=0;                    /* this will be altered by timer interrupt */
    readNewEncoder();

    while (time<tc) ;          /* wait 2 sec */

    readNewEncoder();

    allOffAndZero();

    return;
}  /* end of circumferenceTest */



/* ---------------------------------------------------------------------- *
 * steerTest()                                                            *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  29 October 1997                                          *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      This function computes the actual servo readings for all *
 *               steering motors.                                         *
 * Called from:  user() upon keyboard interaction (type 'w')              *
 * ---------------------------------------------------------------------- */
void steerTest()
{
    unsigned int   *servoControl=(unsigned int *)VME2170;           /* Data Out */
    unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);     /* Analog out */
    unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */
    unsigned char *p;
```

```
    unsigned short bitMask=0x8000;       /* access bit 15 for align  wheel 1 */
    unsigned int wheelSelect=0x00004000; /* select servo for turning wheel 1 */
    int ix,turns,a;

    /* align wheels */
    align();

    /* clear all driving and steering motor counters and the variables */
    clearEncoder(0xff);

    servoOut=(unsigned short*)(VME9210+0x008A); /* Analog out for steering wheel 1  */
    bitMask=0x8000;                             /* access bit 15 for align  wheel 1 */
    wheelSelect=0x00004000;                     /* select servo for turning wheel 1 */

    readNewEncoder();                          /* read all encoders */

    for (ix=0; ix<4; ix++) {

       turns=0;
       *servoOut=testSpeed;           /* set output value for servo first        */
       *servoControl=wheelSelect;   /* turn on selected servo motor           */

       /* turn wheels for a total of 10 turns */
       do {
          while(!(bitMask&*servoStatus));   /* wait until wheel aligned    */
          while(bitMask&*servoStatus);      /* wait until wheel progressed */
          turns++;                          /* one turn completed          */
          if (turns==1)
             timeForTurn[ix]=intCounter;    /* store time (start observing) */
          if (turns==9){
             timeForTurn[ix]=(intCounter-timeForTurn[ix])/8;   /* stop timer */
            *servoOut++=0x0800;             /* speed for final turn  */
          }
       }while (turns<10);

       wheelSelect= wheelSelect<<3;      /* select next servo (motor)        */
       bitMask = bitMask >> 1;           /* select ner xt status align bit   */
    }

    *servoControl=0x00000000;           /* disable (turn off) all wheels    */

    readNewEncoder();

    for (ix=0; ix<4; ix++) radPerDigit[ix]=2.0*PI*10.0/(double)steerDelta[ix];

    return;
} /* end of steerTest */


/* ------------------------------------------------------------------------- *
 * stopTest()                                                                *
 *                                                                           *
 * Environment:  GCC Compiler v2.7.2                                         *
 * Last update:  03 November 1997                                            *
 * Name:         Thorsten Leonardy                                           *
 * Purpose:      This function computes the actual servo readings for all    *
 *               steering motors while the motor speeds are set to zero.     *
 * Called from:  user() upon keyboard interaction (type 's')                 *
 * ------------------------------------------------------------------------- */
void stopTest()
{

    sioOut(0,"Aligning Wheels ...\n\r");

    align();    /* align wheels */

    /* clear all driving and steering motor counters and the variables */
    clearEncoder(0xff);

    readNewEncoder();
    allOnAndZero();

    time=0;
    sioOut(0,"Please Wait a minute ...\n\r");
    while (time<6000) ;   /* wait a minute */
    allOffAndZero();
```

```
    sioOut(0,"Done\n\r");
    readNewEncoder();

    return;
} /* end of stopTest */




/******************************************************************************
  End of motor.c
 ******************************************************************************/
```

APPENDIX D: SHEPHERD PRIMER

This appendix provides essential data and procedures which lead to the findings of the motion parameters that are required to operate SHEPHERD properly. Boxed text will refer to a segment of software code or a command sequence for use in the TAURUS Debugger environment. The focus is on the use of the TUARUS Debugger since this provides a quick way to determine most of the operating parameters.

1. MAIN OPERATING PARAMETERS AND CONVERSION FACTORS

It is sometimes tedious to gather the meat for operating a system. This section strives to provide most of the operating parameters pertaining to the use of SHEPHERD in tabulated form.

Wheel Radius	0.189 m
max. Tire pressure	49.8 psi
Drive Encoder (all Wheels)	$2\ \pi$ radians = 360 * 290 counts
	1 m = 87914 counts
	1 count = 11.37 μm
Wheel 1	digit = 187.20 v [cm/sec] - 26.4
Wheel 2	digit = 187.04 v [cm/sec] - 26.4
Wheel 3	digit = 186.88 v [cm/sec] - 4.8
Wheel 4	digit = 187.20 v [cm/sec] + 8.8
Steer Encoder (all Wheels)	$2\ \pi$ radians $\equiv$ 360 * 256 counts
	1 degree = 256 counts

Table 4.1: Shepherd Operating Parameters in a Nutshell

2. RESET AND READ SHAFT ENCODERS

To find out how the servo readings relate to either the steering and/or the driving, use the following debugger sequence which resets the servo counter for one wheel, drives the wheel and reads the servo counter after steering is done. The same procedures would apply for use with the remaining servo motors.

```
Taurus_Bug>ms ffff610b 04           & clear servo counter for steering wheel 3
Taurus_Bug>ms ffff048e 0800         & set velocity for steering wheel 3
Taurus_Bug>ms fffff00 00100000     & turn on motor for steering wheel 3
                                    & ... after a certain number of revolutions ...
Taurus_Bug>ms fffff00 00000000     & turn off motor for steering wheel 3
Taurus_Bug>ms ffff610b 03           & select control for motor 7 (steer wheel 3)
Taurus_Bug>ms ffff610b 01           &
Taurus_Bug>md ffff6109:1;b          & read least significant byte of 24bit counter
FFFF6109 D3                         & the result
Taurus_Bug>md ffff6109:1;b          & read next byte ...
FFFF6109 C6                         & ... the result
Taurus_Bug>md ffff6109:1;b          & read most significant byte ..
FFFF6109 FB                         & ... the result
Taurus_Bug>                         & the complete counter value in this case is
                                    & 0xfbc6d3 sign-extended (e.g. -276781)
```

3. UP- AND DOWNLOADING DATA FROM TAURUS BOARD

At this time, there is no straight forward routine for data up- and downloading available. Hence, the up- and downloading of data such as waypoints, ... is very tedious. The only way, data can be transferred from or to the TAURUS main memory is via the TAURUSBug options `'du'` for downloading data to the Laptop and `'lo'`. However, data would be made available only in form of the Motorola S-Record format.

To download data from the TAURSU main memory to the Laptop, the Laptop must capture the script sent to the screen to a file (option "T"ext "C"apture on the menu bar). In a second step, output the data to the screen using the folowing command:

```
Taurus_Bug>du0 100000 1000ff 'This is a dump to the screen'
Effective address: 00100000
Effective address: 001000ff
S01F00005468697320697320612064756D7020746F20746865207363726565 6EF1
S21410000001234123412341234123412341234AB
S21410001012341234123412341234123412349B
S21410002012341234123412341234123412348B
S21410003012341234123412341234123412347B
S21410004012341234123412341234123412346B
S21410005012341234123412341234123412345B
S21410006012341234123412341234123412344B
S21410007012341234123412341234123412343B
S21410008012341234123412341234123412342B
S21410009012341234123412341234123412341B
S2141000A012341234123412341234123412340B
S2141000B01234123412341234123412341234FB
S2141000C01234123412341234123412341234EB
S2141000D01234123412341234123412341234DB
S2141000E01234123412341234123412341234CB
S2141000F01234123412341234123412341234BB
S9030000FC
Taurus_Bug>
```

As can be seen above, the data from memory location `0x100000` to `0x1000ff` will be output to the screen and thus captured in the ascii file specified. However, the data will be in the Motorola S-Record format and a parsing program needs to extract the pure data. The parsing program however, needs to know the datatype of the data given to extract the correct information. E.g., extracting data of datatype 'integer' would require a different parsing routine.

As far as the uploading of data is concerned, the datafile must be transferred in the same manner as the SRK program, with the `'LO'` option and described by [1].

4. INTERRUPTS

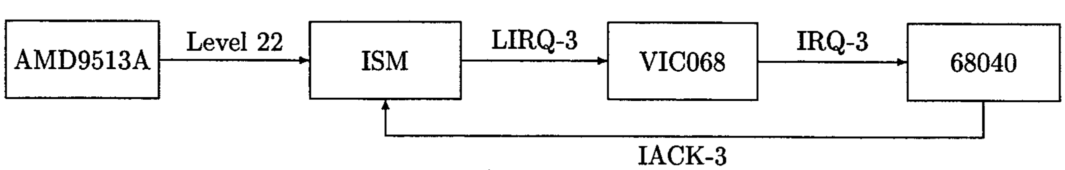
This section describes briefly what type of interrupts are enabled on SHEPHERD.

a. Timer Interrupt

Every 10 ms, a timer interrupt is issued by the on board timing circuit. The interrupt handling routine 'TimerHandler' does the following:

1. increments counter 'intCounter'
   (which may be needed for timing purposes)
2. initiate (software trigger) a block conversion for the A/D-Board AVME9325-5
3. call function 'driver' in file 'movement.c' to execute/handle motion control part

The interrupt is routed through the Interrupt steering mechanism (ISM) to the VIC068 and from there to the 68040 processor in the following way:

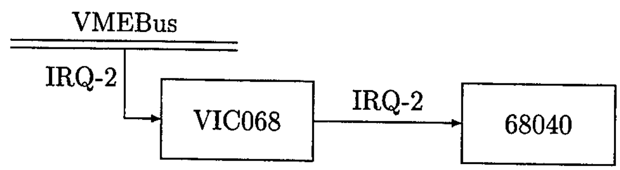


b. A/D-Board Interrupt

Every 10 ms, the timer circuit initiates the start of a block conversion on the A/D-Board. Once this conversion is complete, the A/D-Board AVME9325-5 issues an interrupt to indicate that

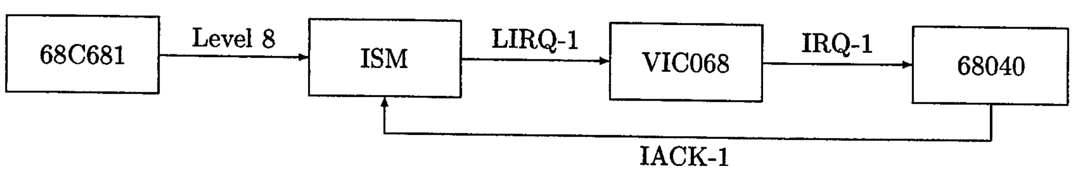
the conversion is complete and data is available to be read from its dual port RAM. The interrupt handler **'handlerVME9325()'** then subsequently calls 'analyzeData' to further analyze/process the data. The interrupt vector number is provided by the Board and set to be 0x0056 which relates to the location of the address for interrupt handling routine at 0x0158 in the interrupt vector table.

As opposed to on-board interrupts, the interrupt from the A/D-Converter VME board is routed directly through the VIC068 to the 68040 processor:



c. Keyboard Interrupt

The overarching framework for user interaction is provided by the routine 'user()' in file 'user.c'. Each time, the keyboard is pressed, an interrupt is issued by the 68C681 on board serial circuit to the 68040 through the ISM and VIC068. The ascii code for the key pressed is then be stored in the variable inPortA and further analyzed by the routine 'user()' in file 'user.c'. The mode flags set in this function will be further processed by functions called during the motion control cycle following each 10ms timer interval. For this interrupt, the interrupt vector number is provided by the DUART and set to be 0x0060 thus giving rise to the location of the interrupt handling routine **inPortAHandler** at 0x0180 in the interrupt vector table.



5. REPRESENTATION OF DOUBLE VARIABLES

According to the M68040 users manual, any double-precision variable is stored in memory as an 8 byte data value in the following form

Since the representation is normalized with the leading (implicit) bit always one we find the relation

Bit 63	= s =	sign bit (1=negative number)
Bit 62..52	= e =	11 bit exponent in the range 0x000 ... 0x7ff
Bit 51..0	= f =	52 bit (13 nibbles) binary decimal (mantissa) in the range 0x0000000000000 ... 0xfffffffffffff

to the real number representation x by

$$x = (-1)^s \, 2^{e-\texttt{0x3ff}} \, (1 + d)$$

with $d = f \cdot 2^{-52}$. As an example, to display the double variable stored in memory location 0x306e8 we issue the following TAURUSbug commands

```
Taurus_Bug>md 306e8:1;d
000306E8 1_3F1_1DF44179E4364
```

The result is conveniently displayed by the monitor such that the elements can be easily identified: s=1, e=0x03f1, f=0x1df44179e4364. Hence, the real number is

$$x = (-1)^1 \, 2^{(\texttt{0x03f1-0x03ff})} \, (1 + \frac{\texttt{0x01df44179e4364}}{\texttt{0x10000000000000}})$$

6. HOW TO RUN SHEPHERD'S WHEELS

Three VME boards account for operating of the wheels, both in steering and driving. These boards are accessible via the VME Bus Port connector P1 and they are:

Board	Function	GCC Access
VME 9210	Analog Output to servos (velocity)	short
VME 2170	Servo Control (on/off)	unsigned int
VME 9421	Servo Status	unsigned short

Shepherd is equipped with a total of eight servo motors: four wheels with driving and turning capability. The setup and software configuration is depicted in Figure (1). In order to operate each one of the motors one has to perform the following steps:

1. Select the angular velocity for the motor by writing a signed short value (16 Bit) to the respective channel (see Figure 1 for the channel assignments) on the VME9210 board (analog Output). E.g. to turn wheel 3 (rear right) one would write

```
*(ffff048e)=(short)velocity;
```

where a positive velocity corresponds to the spin direction as indicated by the arrow in Fig. (1). The well known Right-Hand rule applies for determining the direction of spin.

2. Switch the motor on/off by writing the respective mask to VME2170 at `0xffffff00`. Refer to Fig. (1) for the mask assignment. E.g. to drive wheel 2 (front left) and turn wheel 4 (rear left) simultaneously, one would issue the command

```
*(0xffffff00)=(unsigned int)0x00800020
```

Any combination is allowed, i.e. mask 0x00900000 would turn wheels 3 and 4. Make sure you have set the angular velocities for the wheels you are going to run as outlined in step 1 above!

A word of Caution: for driving wheels 1 (front right) and 3 (rear right) forward, negative values must be written to the VME9210 Board as outlined in step 1.

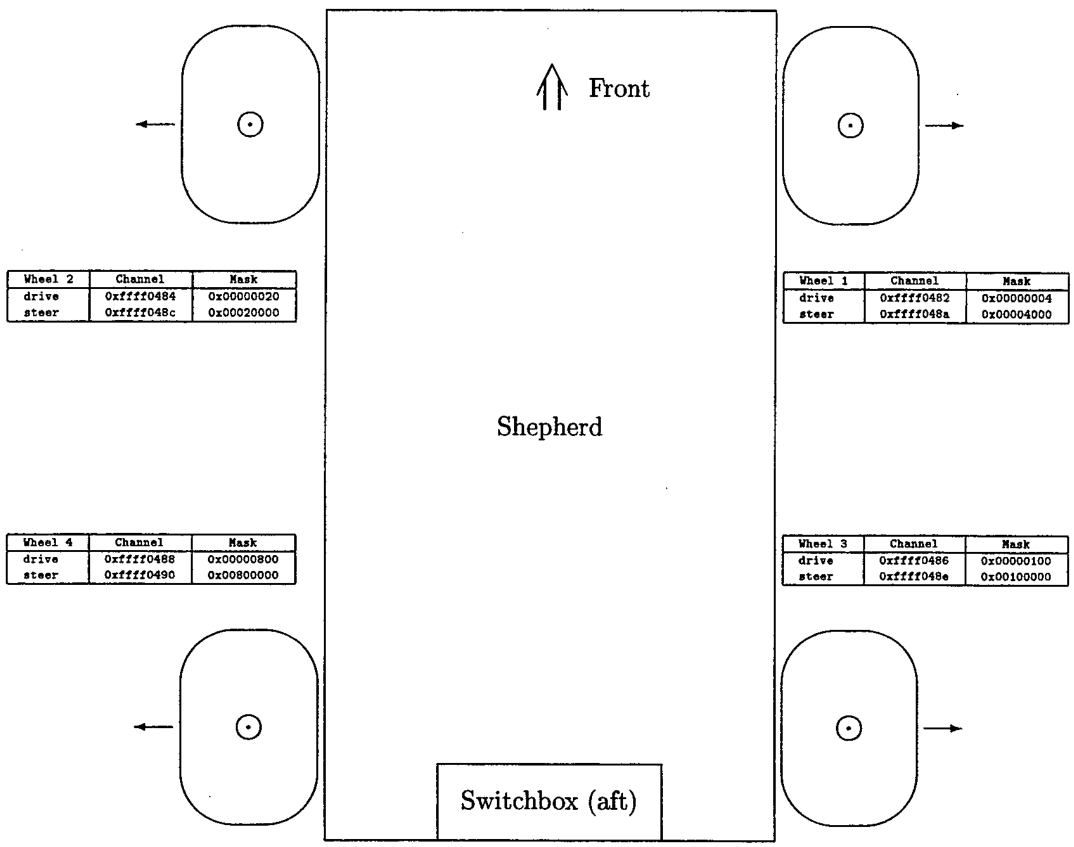


Figure 4.1: Wheel Assignment and Servo Register Addressing (Arrows and Dots at each wheel indicate the rotation of the respective servos if controlled with positive values.

LIST OF REFERENCES

[1] Mays, Edward J. and Reid, Ferdinand A., *Shepherd Rotary Vehicle: Multivariate Motion Control and Planning*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1997

[2] Kanayama, Yutaka, et.al. *Research on a Semi-Autonomous Ground and Aerial Vehicle System for Mine/UXO Detection and Clearing*, Naval Postgraduate School, Monterey, CA, August 1996

[3] *TUARUS 68040/68060 VMEBus Single Board Computer*, User's Manual, Omnibyte Corporation, West Chicago, IL, March 1995

[4] Fowles, Grant R. and Cassiday, George L., *Analytical Mechanics*, Saunders College Publishing, Orlando, FL, 1990

[5] Kaplan, Elliot D., Editor *Understanding GPS, Principles and Applications*, Artech House Inc., Norwood, MA, 1996

[6] Craig, John J., *Introduction to Robotics: Mechanics and Control*, 2nd Edition, Addison-Wesley Publishing Company, Inc., Reading, MA, 1995

[7] Fossen, Thor I., *Guidance and Control of Ocean Vehicles*, John Wiley & Sons, West Sussex, England, 1994

[8] Systron Donner Operating Manual, *MotionPak, Solid State Motion Sensor, Model MP-1, Specification MP-G–CQBBB-100*, Systron Donner Inertial Division, Concord, CA

[9] Systron Donner, *MotionPak, Final Test Data Sheet, Model MP-G–CQBBB-100*, Systron Donner Inertial Division, Concord, CA, 5 November 1996

[10] *ACROMAG Series 9325 High Speed Analog Input Board with RAM*, User's Manual, Acromag Incorporated, Wixom, MI, 1994

[11] Stoer, Josef, *Einführung in die Numerische Mathematik I*, Springer Verlag, Berlin, Germany, 1972

[12] Gerald, Curtis F., and Wheatly, Patrik O., *Applied Numerical Analysis*, 5th Ed., Addison-Wesley Publishing Company, Inc., Reading, MA, 1994

[13] *Selecting Range and Calibrating Voltage Scale Factor with the QFA7000 (Quartz Flexure Accelerometer)*, Information Sheet, Systron Donner Inertial Division, Concord, CA, January 1995

[14] Proakis, John G. and Manolakis, Dimitris G., *Digital Signal Processing. Principles, Algorithms, and Applications*, Prentice Hall, NJ, 1996

[15] Walker, Randy G., *Design and Evaluation of an Integrated, Self-Contained GPS/INS Shallow-Water AUV Navigation System (SANS)*, Master's Thesis, Naval Postgraduate School, Monterey, CA., June 1996

[16] Welch, G. and Bishop, Gary, *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, NC., June 1997 Available at http://www.cs.unc.edu/~welch/kalman/kalman.html

[17] Shultis, J. Kenneth, *LaTeX Notes, Practical Tips for Preparing Technical Documents*, Prentice Hall, Englewood Cliffs, NJ, 1994

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr. Xiaping Yun, Code EC/Yx 2
   Electrical and Computer Engineering Department
   Naval Postgraduate School
   Monterey, CA. 93943

4. Dr. Xavier K. Maruyama, Code PH/Mx 1
   Department of Physics
   Naval Postgraduate School
   Monterey, CA. 93943

5. Chairman 1
   Department of Physics
   Naval Postgraduate School
   Monterey, CA. 93943

6. Dr. Yutaka Kanayama, Code CS/Kz 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA. 93943

7. Thorsten Leonardy 1
   c/o Erichsen
   Am Wiesenbogen 12
   D-24999 Wees
   Germany

8. Streitkräfteamt/Abteilung III 1
   Fachinformationszentrum der Bundeswehr
   Friedrich-Ebert-Allee 34
   53113 Bonn
   Germany